UCA Universidad de Cádiz

Universidad de Cádiz

Proyectos fin de carrera de Ingeniería Técnica Industrial . Electrónica Industrial

Centro: ESCUELA POLITÉCNICA SUPERIOR DE ALGECIRAS

Titulación: INGENIERÍA TÉCNICA INDUSTRIAL . ELECTRÓNICA INDUSTRIAL

Titulo: Medidor de energía eléctrica microcontrolado

Autor: Fco. Javier Pedrosa Ruiz

Fecha: Junio 2011

Trabajo Fin de Carrera - Medidor de energía eléctrica microcontrolado

Autor: Francisco Javier Pedrosa Ruiz
Titulación: I.T.I. Electrónica Industrial

Contenido

1. JUSTIFICACION Y ANTECEDENTES

La medición de energía eléctrica que se efectúa mediante medidores o contadores resulta de interés para calcular la cantidad de energía que la compañía suministradora debe facturar a los consumidores. Antiguamente estos dispositivos eran grandes y costosos, pero el avance de la tecnología ha hecho posible que estos dispositivos puedan fabricarse económicamente y con un error en la medida considerablemente aceptable.

El presente trabajo fin de carrera consiste en el diseño e implementación de un medidor electrónico de parámetros asociados a la calidad de la energía, empleando software de distribución gratuita e intentando buscar el método más económico y eficaz de implementar los componentes electrónicos. El sistema está basado en el circuito integrado ADE7759 de Analog Devices, el cual es capaz de medir potencia y energía monofásica (entre otras magnitudes), contando con recursos internos para el muestreo de las señales de tensión y corriente, filtrado y compensación de errores. La implementación discreta del dispositivo se apoya en el diseño recomendado por la empresa Analog Devices en las notas de aplicación que se adjuntan en el anexo 5 Notas de aplicación

Este dispositivo no solo sirve para la medición del consumo energético de los dispositivos eléctricos; en la actualidad existe una fuerte tendencia al diseño de dispositivos integrados con cierto grado de inteligencia o prestaciones (smart instruments) asociados a la calidad de la energía eléctrica y al uso eficiente de esta. En este trabajo se emplean dispositivos de última generación con prestaciones como la medida de corriente eficaz, tensión eficaz, potencia activa, potencia reactiva y factor de potencia, además del consumo energético

El trabajo se divide en cuatro bloques; el primero describe los materiales empleados para la realización de este trabajo, tanto el material fungible como el software empleado para documentar todas las experiencias. También se incluye en este apartado una lista de todos los componentes utilizados y el coste de estos. En el bloque "Métodos" se exponen los fundamentos y consideraciones a la hora de diseñar cada función del

medidor. El apartado "Resultados" describe la forma de implementar los dispositivos, tanto físicamente en la placa de prototipado como el algoritmo programado para Arduino. Finalmente se realiza una discusión y valoración de los resultados obtenidos. Todos los programas realizados durante este trabajo se incluyen en el CD de este trabajo, así como una hoja de Excel para calcular diferentes coeficientes.

2. OBJETIVOS

- Implementación en una placa de prototipado del circuito integrado ADE7759 con un microcontrolador ATMEGA328 contenido en una placa Arduino. El circuito integrado ADE7759 se alimentará desde Arduino y la comunicación entre ADE7759-ATMEGA328 se realizará mediante un bus SPI. En base a los resultados derivados de las experiencias realizadas en este trabajo se decidirá si es viable construir el circuito en una placa circuito impreso. El diseño debe considerar en todo momento que el sistema debe permitir la flexibilidad de desarrollo por parte de terceros, para que este pueda ser permanentemente mejorado sin ulteriores inversiones

- Programar los procedimientos y funciones necesarios en el entorno Arduino para que el sistema implementado pueda realizar lecturas de tensión eficaz, corriente eficaz y potencia activa a la frecuencia de la red eléctrica de España (50Hz), con un error inferior al 1%. El sistema de medida incorporará una función de acumulación de energía consumida, y también se programarán algoritmos para el cálculo de otros parámetros derivados de los anteriores, como la potencia aparente o el factor de potencia

- El dispositivo final se alimentará mediante USB o desde la propia red eléctrica en la que se realiza la medida. Los datos se presentaran por una pantalla, preferiblemente instalada en el circuito como un display LCD o similar, aunque siempre hay que considerar que el dispositivo se puede comunicar por el puerto serie con un PC, permitiendo así graficar los datos en una pantalla. El dispositivo mostrará en el LCD un menú con las funciones programadas (corriente RMS, Tensión RMS, Potencia Activa…) por el que el usuario podrá moverse con cuatro botones pulsadores.

3. MATERIALES

3.1 SOFTWARE

- **FRITZING**

Fritzing es una aplicación para la edición de esquemas de conexión para proyectos de electrónica. La versión utilizada es la 0.5.4b, es decir, actualmente se encuentra en fase alfa, por lo que no es un programa que se pueda considerar acabado. Aún así tiene muchas características que la convierten en una aplicación muy interesante tanto para estudiantes de electrónica como para profesionales. Esta aplicación está pensada principalmente para organizar proyectos con Arduino de una forma clara y mediante una interfaz muy sencilla. Cada proyecto que se cree contiene tres vistas principales (protoboard, esquema y PCB) organizadas en pestañas:

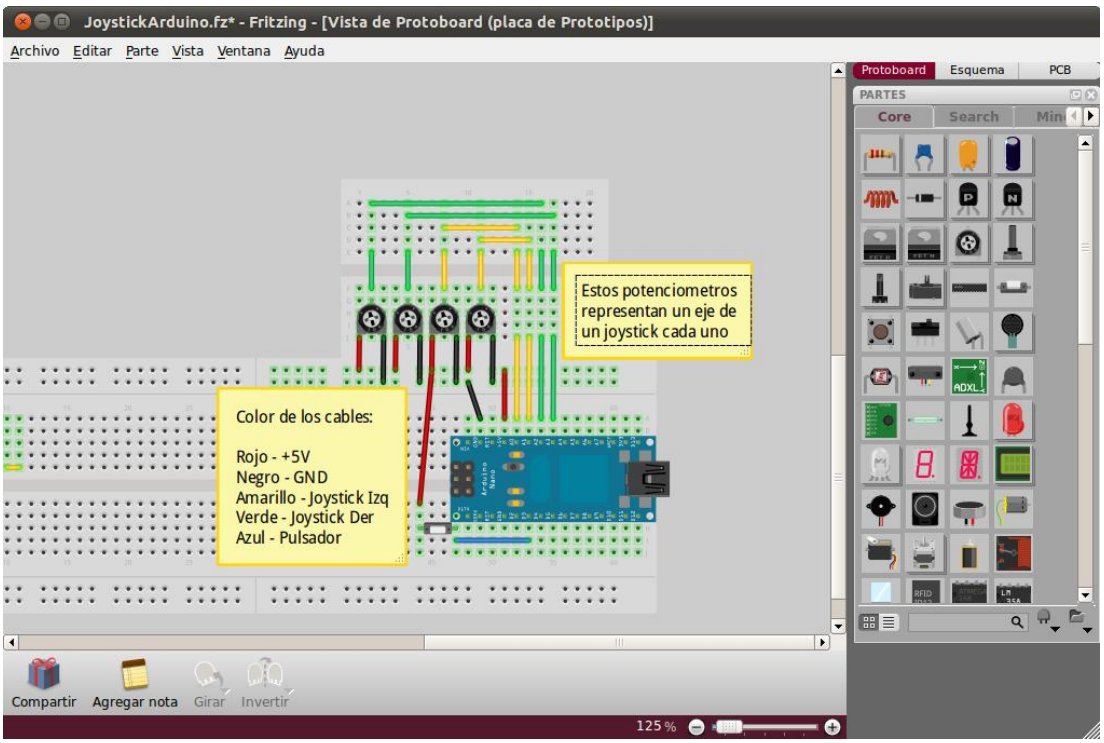


Imagen 3.1.2 – Vista de la aplicación

La "Vista Protoboard" está diseñada para mostrar una visión realista de la implementación del proyecto. Las conexiones se realizan arrastrando los componentes

necesarios y uniéndolos entre sí para formar el circuito. También se pueden añadir notas para aclarar las partes del diseño.

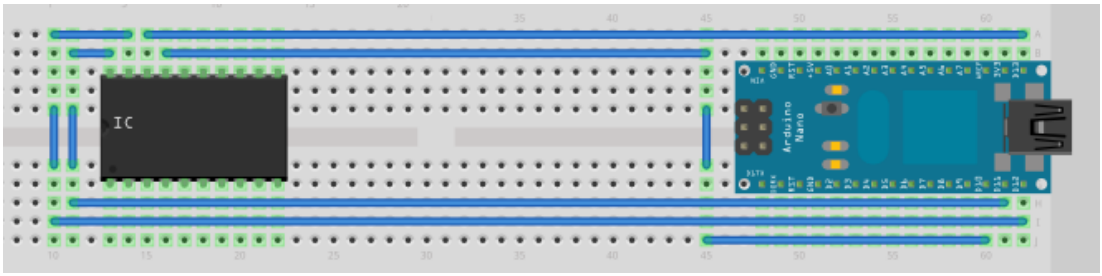


Imagen 3.1.2 – Vista de un circuito implementado en protoboard

En la vista del circuito PCB, o "layout", se indica donde van a ir situados los componentes dentro de la placa de circuito impreso. Se puede modificar tanto el tamaño como la complejidad de conexiones en función de las necesidades. Fritzing tiene una herramienta de autorruteo para realizar las pistas de cobre, aunque esta opción no es muy eficaz, siendo conveniente hacerlo a mano cuando sea posible:

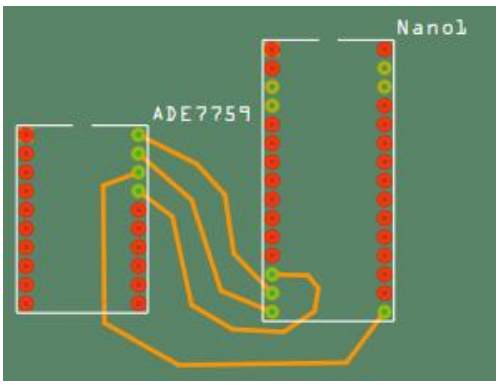


Imagen 3.1.3 – Vista de layout

La vista del esquemático ofrece una forma abstracta de los componentes y las conexiones. Esta vista resulta muy útil para comprobar que las conexiones que realizamos en la vista anterior son correctas.

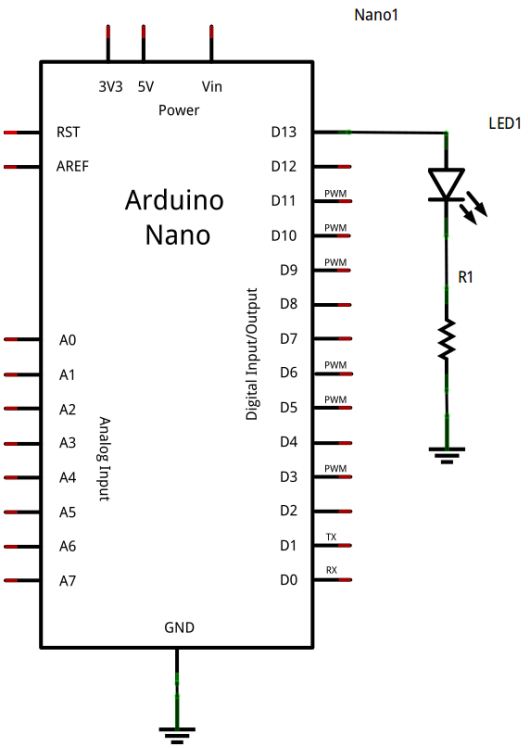


Imagen 3.1.4 - Vista del esquema eléctrico

En cualquiera de las vistas podemos exportar en todo momento nuestro proyecto en diferentes formatos (PDF, JPG, SVG…etc.) En futuras versiones se podrá exportar el circuito a otras aplicaciones, como Eagle para realizar simulaciones.

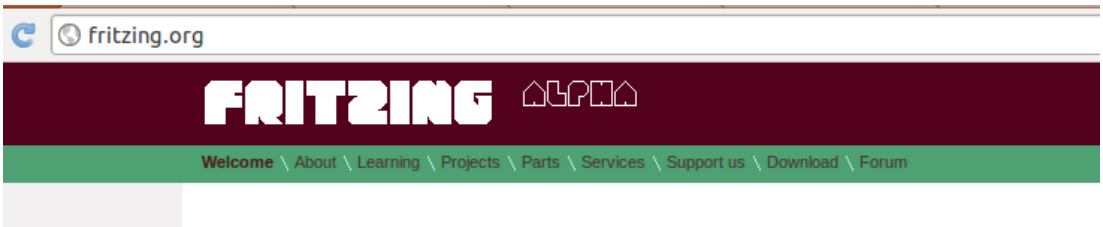


Imagen 3.1.4 – Web oficial de Fritzing

El programa tiene licencia GPL para el código, y Creative Common para el contenido, es decir, puede obtenerse y utilizarse gratuitamente Además cuenta con un sitio web www.fritzing.org complementario que ayuda a compartir y discutir bosquejos y experiencias y a reducir los costos de fabricación.

- **ENTORNO DE PROGRAMACIÓN ARDUINO**

Arduino es una plataforma de hardware libre basada en una sencilla placa de entradas y salidas simple y un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring. Arduino se puede utilizar para desarrollar objetos interactivos autónomos o puede ser conectado a software del ordenador (por ejemplo: Macromedia Flash, Processing, Max/MSP, Pure Data). Las aplicaciones que nos ofrece Arduino son múltiples, y dependerá de nuestra imaginación. Mediante sensores podemos crear aplicaciones sencillas enfocadas a la docencia para estudiantes de electrónica, proyectos más elaborados para la industria o incluso sistemas dirigidos simplemente al ocio.

Imagen 3.1.5 - Vista del entorno de programación arduino

Las plataformas Arduino están basadas en los microcontroladores ATmega168, ATmega328, ATmega 1280 y ATmega 8 y otros similares, chips sencillos y de bajo coste que permiten el desarrollo de múltiples diseños. En el anexo 3 – Documentación de Arduino hay más información técnica del dispositivo.

Es posible utilizar otros lenguajes de programación y aplicaciones populares en Arduino porque este se comunica mediante la transmisión de datos en formato serie, que es algo que la mayoría de los lenguajes anteriormente citados soportan. Para los que no soportan el formato serie de forma nativa, es posible utilizar software intermediario que traduzca los mensajes enviados por ambas partes para permitir una comunicación fluida. Es bastante interesante tener la posibilidad de interactuar Arduino mediante esta gran variedad de sistemas y lenguajes puesto que dependiendo de cuales sean las necesidades del problema que vamos a resolver podremos aprovecharnos de la gran compatibilidad de comunicación que ofrece.

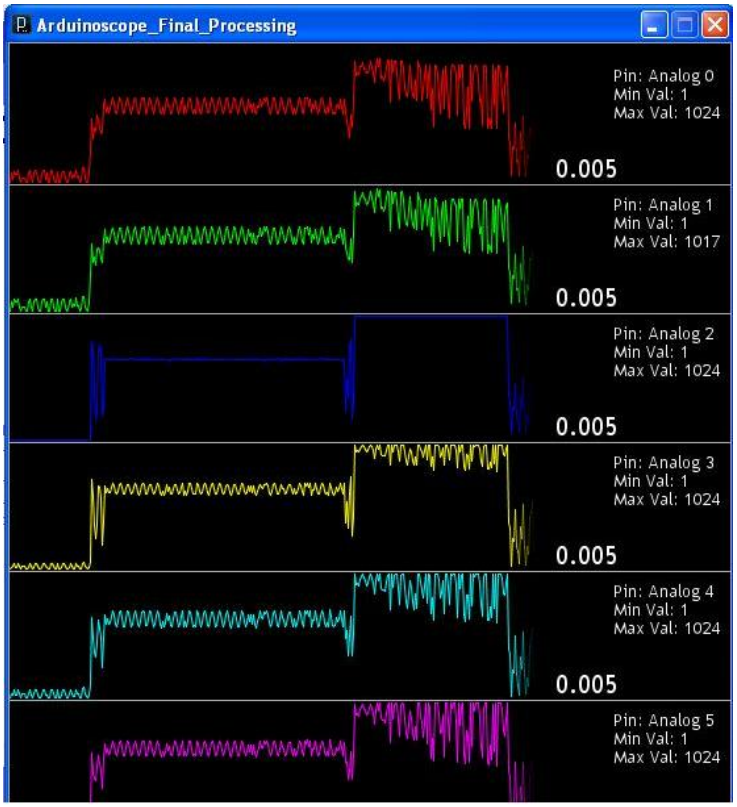


Imagen 3.1.6 - Comunicación Arduino – Processing mediante el puerto serie

El entorno de desarrollo integrado tiene licencia para la distribución y uso de forma gratuita, y se puede descargar fácilmente en internet. Al ser open-hardware, tanto su diseño como su distribución es libre. Es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia.

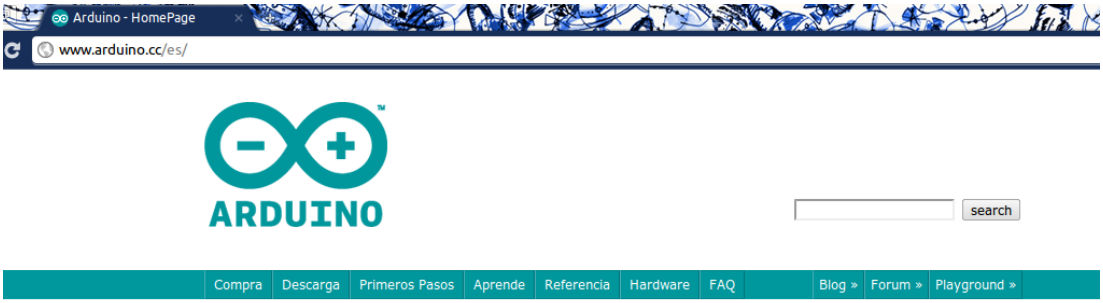


Imagen 3.1.7 – Pagina de Arduino

Otra faceta de Arduino es la comunidad de usuarios. En la página principal de Arduino se pueden encontrar foros de ayuda, explicaciones sobre el funcionamiento, proyectos que la gente comparte…etc. La filosofía de los diseñadores se basó en la creación de un dispositivo para implementar circuitos de una forma simple y económica, intentando reducir la denominada "brecha digital". Por ellos se puede encontrar los esquemas de las placas, los fotolitos y tutoriales sobre cómo se construye. El precio de una placa Arduino ensamblada no supone gran beneficio para el vendedor, es decir, se venden prácticamente a precio de fábrica (coste de los componentes + proceso de ensamblaje + gastos de envío).

3.2. HARDWARE

- **Lista de componentes y coste**

Componente	Valor/Potencia/Tolerancia	Precio	Cantidad
Arduino	1 Arduino Nano	40.80 €	1
ADE7759	1 Circuito	3.4 €	1
Botón Pulsador	Tipo PCB-2Pin/ 4 und.	0,60 €	4
Resistencia	10kΩ / ⅛W / 1%	0.055	1
Resistencia	510kΩ / ⅛W / 1%	0.055 €	1
Resistencia	1kΩ / ⅛W / 1%	0.055 €	4
Resistencia	24Ω / ⅛W / 1%	0,055 €	2
Condensador	100nF / 50V / 10%	0,065 €	2
Condensador	33nF / 50V/ 10%	0,086 €	4
Condensador	10µF / 50V / 20%	0,064 €	3
Condensador	22pF / 50V / 20%	0,318 €	2
Oscilador	3579545 Hz	0,36 €	1
Transformador de Corriente	1:1000 / 60A /	2,264 €	1
Filtro de ferrita	Filtro para montaje en PCB	0,197 €	3
Potenciómetro	10k	0,364 €	1
Adaptador	SSOP24 – DIP24	0,60 €	1
TOTAL	Coste con Arduino	58,151 €	
	Coste sin Arduino	17,715 €	

- **ADE7759**

ADE7759 es un circuito integrado para la medición de potencia activa y energía acumulada con una interfaz serie y una salida de pulsos. El dispositivo incorpora dos ADC de segundo orden sigma-delta, un integrador digital (en el canal 1), la circuitería referencia para los convertidores, sensor de temperatura, y todo el procesamiento de señal necesario para realizar la medición de potencia activa y energía.

El integrador digital del chip permite la conexión directa de sensores de variación de corriente, como una bobina Rogowski. Además elimina la necesidad de un integrador analógico externo y proporciona una excelente estabilidad a largo plazo. El integrador se puede desconectar si se utilizan otro tipo de sensores convencionales como un transformador de corriente. ADE7759 contiene un registro de muestreo de forma de onda y un registro de energía activa capaz de acumular al menos la integración de la potencia activa a plena carga durante 11,53 segundos. Los datos se leen a través de la interfaz serie. ADE7759 también proporciona una salida de pulsos (CF) con una frecuencia que es proporcional a la potencia activa.

Además de la información de la potencia activa, ADE7759 también ofrece varias características para la calibración del sistema, como por ejemplo, corrección de offset del canal, calibración de fase y la posibilidad de compensar desvíos en la medida final. El dispositivo también incorpora un circuito para la detección de caída de tensión de corta duración SAG. El valor umbral de voltaje y la duración (en número de semiperiodos) de la caída son programables por el usuario. La salida ($\overline{SAG}$) pasa a nivel bajo cuando se produce un evento de este tipo.

La salida de cruce por cero (ZX) presenta una señal sincronizada con el punto de cruce cero de la tensión de la línea de entrada (canal 2). Esta señal puede utilizarse para obtener información de temporización o de la frecuencia de la línea de entrada. La señal también se utiliza para controlar el número de ciclos de la línea de entrada en los que se acumula energía. La acumulación de ciclos de energía de línea permite un cálculo rápido y preciso de la energía, y es una función especialmente útil durante la calibración.

La salida de solicitud de interrupción se activa a nivel bajo. Tras una interrupción, el registro de estado de interrupción indica la naturaleza de esta, y permite al microcontrolador gestionar el evento que produce la solicita. El ADE7759 está disponible en una versión SSOP de 20-pin.

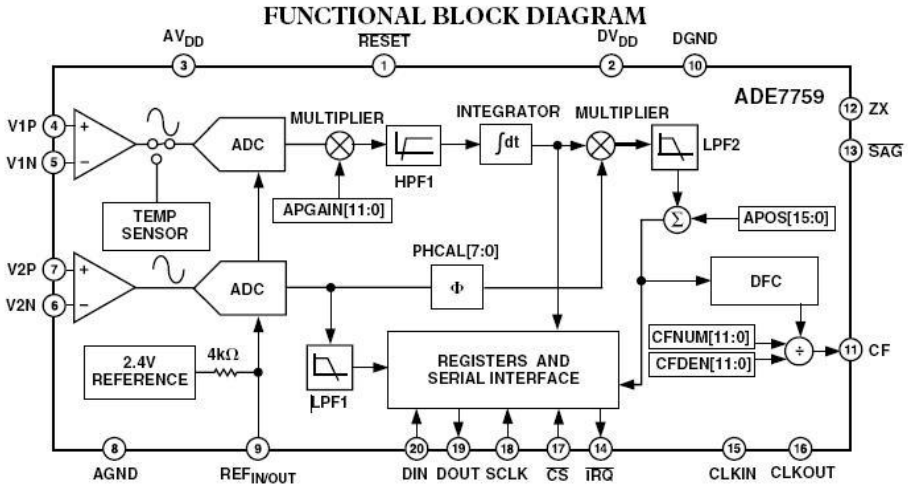


Imagen 3.2.1 – Diagrama de bloques de ADE7759

- **RED DE ENTRADA AL CANAL 1**

El transformador de corriente (TC) usa el principio de un transformador para convertir la alta corriente primaria a una corriente secundaria más pequeña. El TC es común entre los medidores de energía de estado sólido de alta corriente. Es un dispositivo pasivo que no necesita circuitos adicionales de control. Adicionalmente, el TC puede medir corrientes muy altas y consumir poca potencia. Sin embargo, el material ferrítico usado en el núcleo se puede saturar cuando la corriente primaria es muy alta o cuando hay un componente importante de DC en la corriente. Una vez magnetizado, el núcleo contendrá histéresis y su precisión se degradará a menos que éste se desmagnetice de nuevo.

Esta disposición del sensor proporciona aislamiento si la tensión entre líneas difiere más de 300V. Junto con este aislamiento requerido, el CT permite una forma fácil, fiable y efectiva de combinar las corrientes en ambas fases. La siguiente imagen muestra la aplicación usada en este diseño para un medidor monofásico de 2 cables.

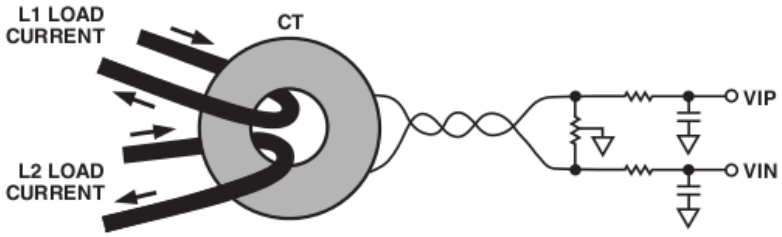


Imagen 3.2.2 – Conexión del TC a ADE7759

La resistencia de carga se selecciona para dar el rango de entrada apropiado. A continuación se el circuito equivalente del sensor y la característica del sensor empleado:

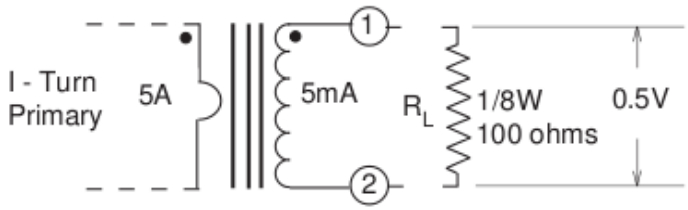


Imagen 3.2.3 – Circuito equivalente del TC

Electrical Specifications @ 20°C ambient

Electrical Specifications	
Primary Current	5A nom., 60A max.
Turns Ratio	1000:1 nominal
Volt per Amp Ratio at 5A for 100 ohm load	0.100 V/A
Volt per Amp Ratio at 0.5A for 100 ohm load	0.096 V/A
DC Resistance at 20°C	41.8 ohms
Dielectric Withstanding Voltage (Hi-pot)	4KVrms

Imagen 3.2.4 – Especificaciones técnicas del sensor TC

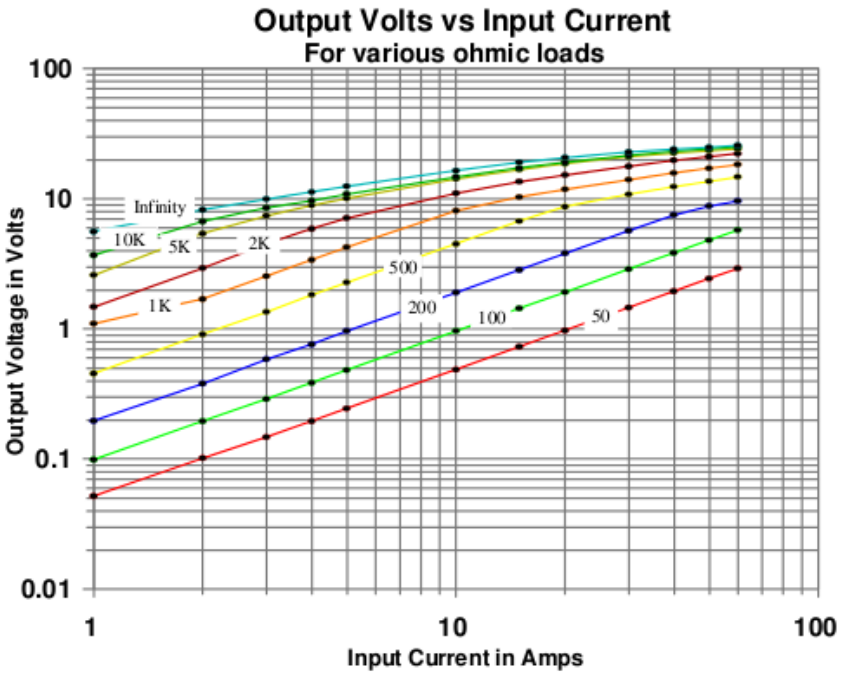


Imagen 3.2.5 – Característica del sensor de corriente

Como se puede observar en la gráfica, la tensión de salida está determinada por la resistencia de carga, y esta a su vez determinará el rango de medida. Para este trabajo se ha usado una resistencia de carga de 50Ω. El valor de la resistencia de carga depende de la máxima corriente se quiera medir y del número de vueltas del secundario del transformador. Para medir una corriente máxima de 10 Amperios se necesita una resistencia de carga de 50 Ω para tener 0,5 Voltios a la salida (valor máximo en el convertidor). El valor mínimo de corriente es 1 Amperio, que con la carga anterior supondría 0,05Voltios

- **RED DE ENTRADA DEL CANAL 2**

Las muestras de tensión se toman mediante un divisor de tensión para disminuir la tensión de línea La topología de la red es de modo que la fase entre el canal 1 y el canal 2 se conserva. Como se puede ver en la Figura 4, el codo de -3dB en la frecuencia de esta red está determinada por R8 = 1kΩ y C8 = 33nF. Las resistencias R7 y R6 forman una resistencia equivalente de 510kΩ

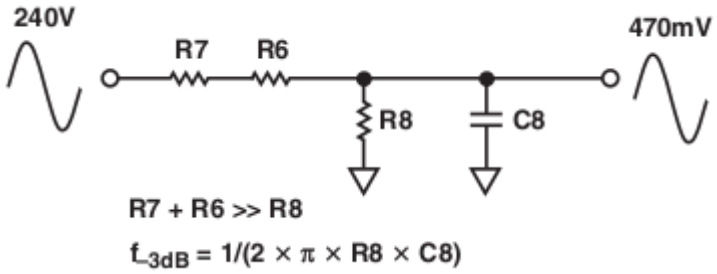


Imagen 3.2.6 - Red de entrada del canal de tensión

Este medidor también puede usarse en aplicaciones a 120V. La red de atenuación no necesita cambiar ya que el canal de tensión puede modificarse internamente en el ADE7759. El registro de ajuste de ganancia PGA del ADE7756 está configurado para acomodar la tensión de entrada de 120V. Esto permite al usuario programar la ganancia de tensión del canal a través de software, manteniendo el SNR. Debe notarse que conmutando el medidor para operar a 120V, el transformador de alimentación debe estar configurado adecuadamente para operar a esa tensión de línea.

Como la función de transferencia del ADE7759 es extremadamente lineal, todo lo necesario para calibrar el medidor es un punto de calibración (IREF) para un factor de potencia unitario. Si se toman las precauciones necesarias al diseñar la etapa, no será necesaria ninguna calibración para bajos factores de potencia (PF = 0.5)

4. MÉTODOS

4.1. ADAPTAR FORMATO DE COMPONENTES

Un problema que aparece a la hora de implementar circuitos con componentes de última generación, es que estos se presentan empaquetados en diferentes formatos, los cuales no siempre son adecuados para trabajar con placas de prototipado. Durante la implementación del medidor ha habido un componente con este problema; el circuito ADE7759 integrado en un encapsulado SSOP-20:

Imagen 4.1.1 - Circuito integrado en formato SSOP-20

La solución consiste en un adaptador de formatos. Hay varias versiones en el mercado, y es recomendable comprar siempre que sea posible el que tenga más conexiones, es decir, el que tenga más pistas. Esto es porque si tenemos un adaptador de SSOP8 no podemos usarlo con un integrado SSOP20, sin embargo si se puede hacer inversamente. En la siguiente imagen se muestra un adaptador de plástico con una pestaña para asegurar que el circuito integrado hace contacto con las pistas:

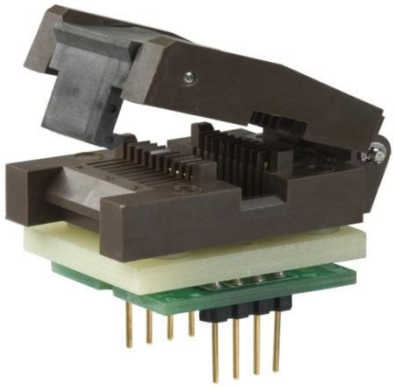


Imagen 4.1.2 - Adaptador SSOP – DIP

La ventaja de este tipo de adaptadores es que no es necesario soldar el circuito integrado, por lo que puede usarse en múltiples circuitos. Sin embargo el principal inconveniente es el precio. Otra solución más económica para poder trabajar en la placa de prototipado es la PCB sin el adaptador de plástico, siendo necesario soldar el circuito integrado.

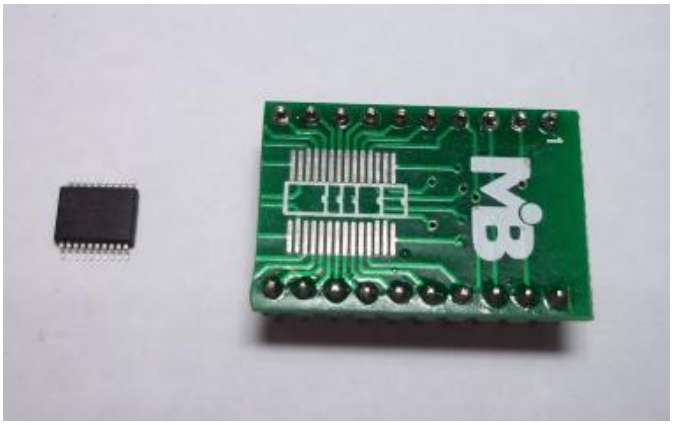


Imagen 4.1.3 - PCB – Adaptador de formato SSOP-20 a formato DIP-20

Este adaptador ha sido adquirido en el portal de compras EBAY, a un coste de 0,6€ la unidad, estando incluido en el precio los conectores DIP y los gastos de envío. El adaptador es de SSOP24 a DIP24, por lo que soban 4 pistas de conexiones. Para reducir el espacio que ocupa el componente en la placa se cortado la parte que corresponde a las conexiones que no se usan en la placa.

Aunque pueda parecer una tarea laborioso la de soldar el circuito integrado en la PCB, es un ejercicio que se puede realizar fácilmente con buenos resultados sin utilizar materiales costosos; para soldar este circuito se ha usado unas pinzas, estaño, un soldador de estaño y flux. El procedimiento consiste en los siguientes pasos:

1. Aplicar Flux en las pistas del adaptador PCB

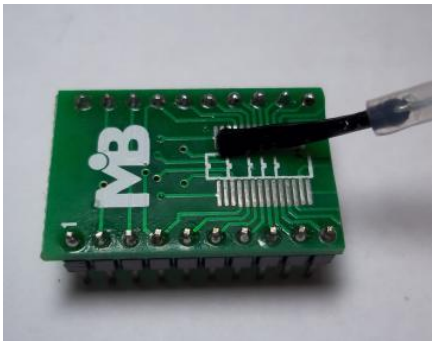


Imagen 4.1.4 – Aplicando flux en las pistas

2. Estañear las pistas en contacto con el circuito integrado

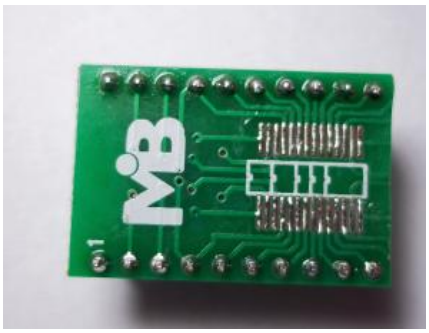


Imagen 4.1.5 – Adaptador con pistas estañeadas

3. Situar el circuito integrado sobre las pistas y aplicar Flux al circuito integrado y las pistas

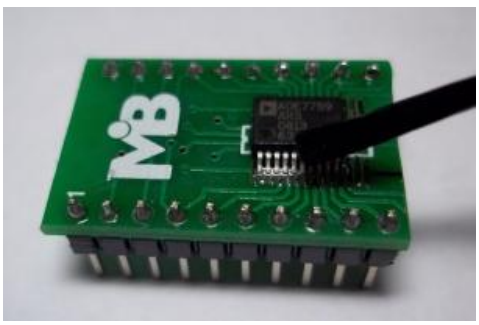


Imagen 4.1.6 – Aplicando flux al CI y a las pistas del adaptador

4. Utilizando las pinzas, colocar el circuito integrado en su posición y aplicar presión verticalmente para que no se mueva. Con el soldador, aplicar calor en las esquinas para que el circuito integrado se quede fijado en su sitio

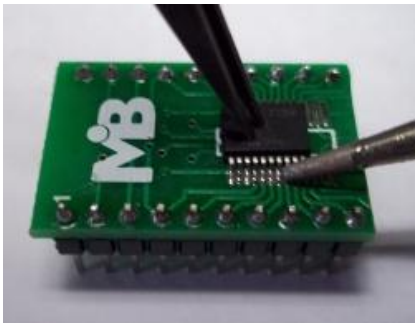


Imagen 4.1.7 – Presionar verticalmente y soldar las esquinas

5. El siguiente paso es aplicar calor a las patillas del integrado para que se queden pegadas a las pistas. En caso de que haya exceso de estaño, se retira moviendo la punta del soldador por las pistas desde el CI hacia fuera. La siguiente imagen muestra el circuito a medio soldar, se puede ver cómo están soldados los pines 1, 3, 5 y 6. El resto, se puede apreciar que aun están despegadas de la pista de la PCB:

Imagen 4.1.8 – CI soldado al adaptador

6. Una vez que el circuito este soldado a la placa, comprobar que hay continuidad entre el pin de conexión de la PCB y cada patilla del integrado, y también la continuidad entre pines para asegurar que no hay cortocircuito. De esta se puede asegurar que el circuito se ha soldado correctamente.

4.2. ALIMENTACIÓN DEL SISTEMA

La placa Arduino puede ser alimentada vía USB o con una fuente de alimentación externa. La fuente de alimentación externa puede consistir en un adaptador AC-DC o una batería. En caso de conectar el cable USB y una fuente externa, Arduino seleccionará automáticamente la procedencia de la alimentación evitando cualquier conflicto. Cuando el sistema se alimenta por USB, existe una limitación de corriente de 500mA, además de una protección para cortocircuitos

El adaptador se puede conectar en el jack de conexión que traen algunas placas como Arduino Duemilanove, o conectar directamente mediante las líneas marcadas como Gnd y Vin en las conexiones de potencia con la etiqueta POWER.

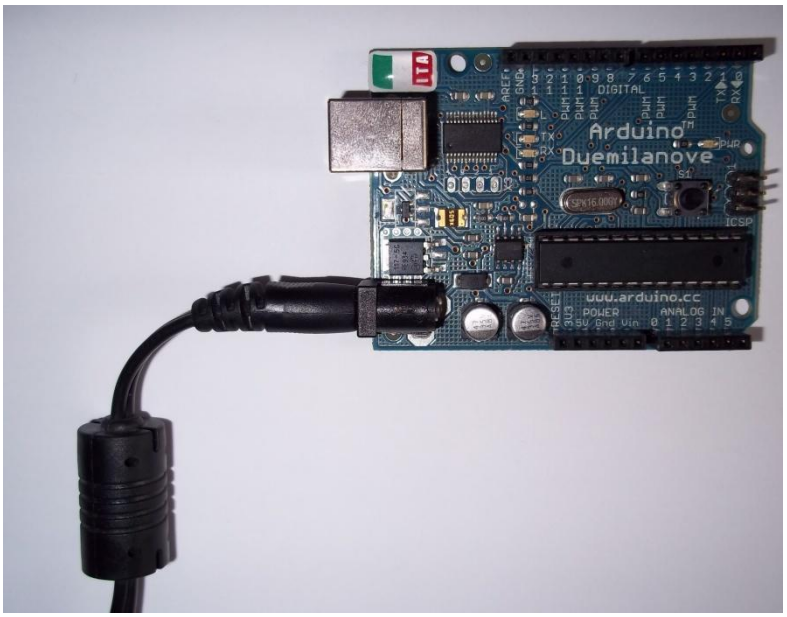


Imagen 4.2.1 - Arduino Duemilanove con cable de alimentación

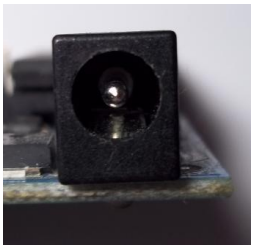


Imagen 4.2.2 - Jack de conexión de placa Arduino Duemilanove

Imagen 4.2.3 - Adaptador AC – DC compatible con la placa Arduino

La placa puede operar con una alimentación externa de entre 6 y 20 voltios. Si se alimenta con menos de 7V, la salida de 5V podría presentar menos de esta tensión y la placa podría presentar un comportamiento inestable. Si se usan más de 12V, el regulador de tensión podría recalentarse y dañar la placa. El rango recomendado está entre 7 y 12 voltios. Los pines de alimentación son los siguientes:

- **VIN.** Línea de entrada de tensión a la placa del Arduino cuando se usa una fuente de alimentación externa. Se puede aplicar tensión a través de este pin, o, si se está alimentando la placa mediante el jack de conexión, acceder a ella a través de él.
- **5V.** La fuente de alimentación regulada que alimenta al microcontrolador y otros componentes de la placa.

- **3V3.** Alimentación de 3,3 voltios generado por el chip FTDI de la placa. La máxima corriente de salida es 50 mA.
- **GND.** Tierra.

Uno de los objetivos de este trabajo está directamente relacionado con la forma de alimentar el sistema. El dispositivo final debe poder alimentarse de la misma red en la que se está midiendo, o mediante un cable USB.

Para alimentar el sistema desde la red, es necesario un adaptador formado por un transformador AC-AC, un puente de diodos y un filtro para reducir el rizado. Una limitación a la hora de elegir este dispositivo es la potencia que consume el sistema que queremos alimentar. La aplicación que se diseña en este trabajo consume poca potencia; 35mW como máximo ADE7759, el display LCD consume habitualmente 7,5mW y la placa Arduino tiene un consumo variable en función de la operación que esté haciendo. Es importante considerar que una misma placa Arduino se puede usar para hacer diferentes proyectos; por esto es interesante tener una fuente de alimentación externa que este sobredimensionada, con vista a que si la aplicación lo requiere podamos disponer de la máxima potencia posible.

Para alimentar ADE7759 se puede usar la salida de alimentación regulada de 5V que incluye Arduino. Según la recomendación del fabricante de ADE7759 las entradas AVdd y DVdd deben conectarse a una tensión de alimentación de 5V +- 5% y en una hay que conectar dos condensadores en paralelo, uno electrolítico de 10uF y otro cerámico de 100nF. En el caso de AVdd los condensadores de conectan entre esta señal y AGND, y para DVdd entre esta y DGND.

4.3. REDUCCIÓN DE RUIDO EN LAS SEÑALES DE ENTRADA

Como consecuencia de interferencias entre los canales del PCB o en el propio CI, aparece ruido en las señales de entrada del medidor. La siguiente imagen muestra dos señales sinusoidales, que son la misma señal pero una está afectada por ruido y la otra no:

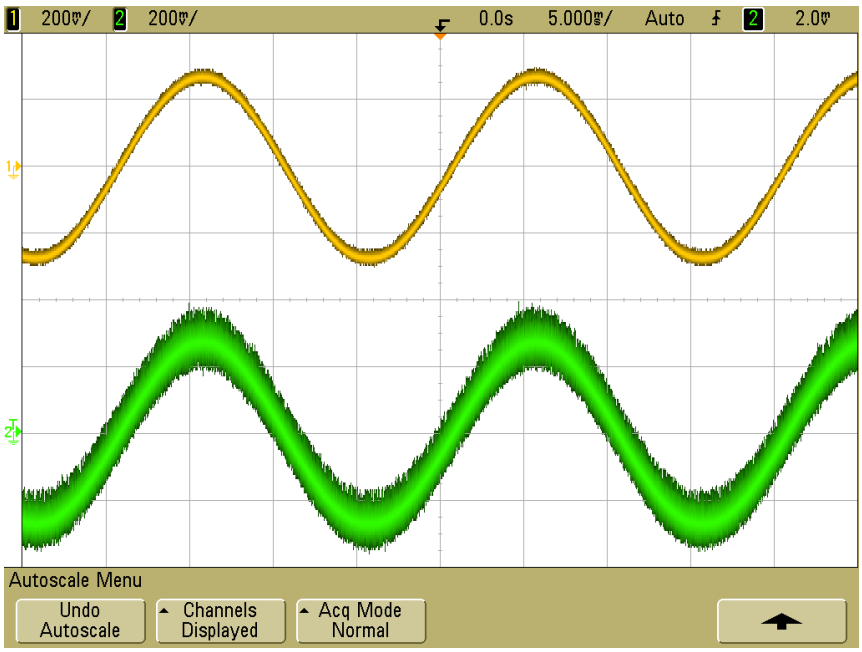


Imagen 4.3.1 Señal del generador (amarillo) y la misma señal en la protoboard (verde)

La señal amarilla es una señal sinusoidal de 50 Hz que está siendo generada por un generador de señales. Esta señal se conecta directamente al canal de entrada del osciloscopio mediante un cable apantallado para protegerlo de interferencias. La señal verde se obtiene conectando al canal del osciloscopio una sonda, y midiendo la tensión en la entrada del PGA de ADE7759. Este punto está también conectado al generador de señales. De esta forma se evidencia el ruido que distorsiona la señal a medir. Este ruido no puede ser eliminado completamente, pero si se puede reducir haciendo uso de varios métodos. Como se observa en la imagen, este ruido se caracteriza por ser de una frecuencia mucho mayor que la de la señal, por lo que los métodos de reducción de ruido tratan de reducir estas componentes de alta frecuencia y dejar intactas las de baja frecuencia.

Un método para reducir el ruido consiste en usar núcleos de ferrita en los puntos en los que el circuito de medida se conecta con el "exterior", es decir, en los puntos de conexión del circuito con la red. La siguiente imagen muestra el filtro que se ha usado:

Imagen 4.3.2 – Filtro de ferrita para protoboard

Este componente presenta un alto valor óhmico (relativamente alto) para señales de alta frecuencia, y bajo valor para señales de baja frecuencia. A continuación se muestra la curva característica del componente y el circuito equivalente:

Imagen 4.3.3 – Circuito equivalente del filtro de ferrita

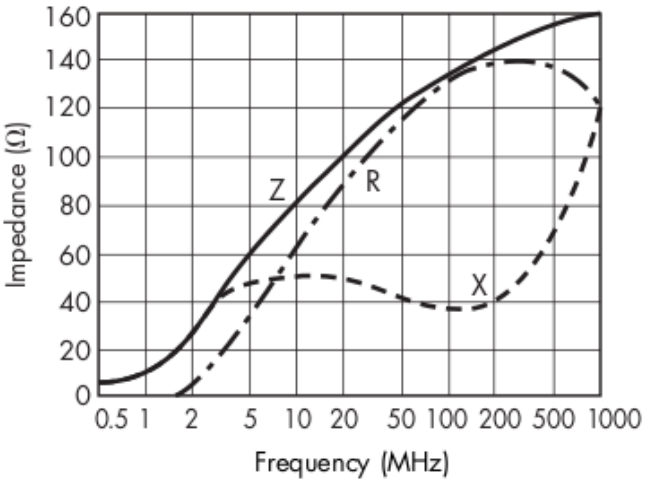


Imagen 4.3.4 – Curva característica del comportamiento del dispositivo

La siguiente imagen muestra las mismas señales que en la imagen anterior, pero esta vez, se han colocado filtros de ferrita entre la conexión del generador de señales y la protoboard. Se puede apreciar como disminuye el ruido de la señal de manera considerable:

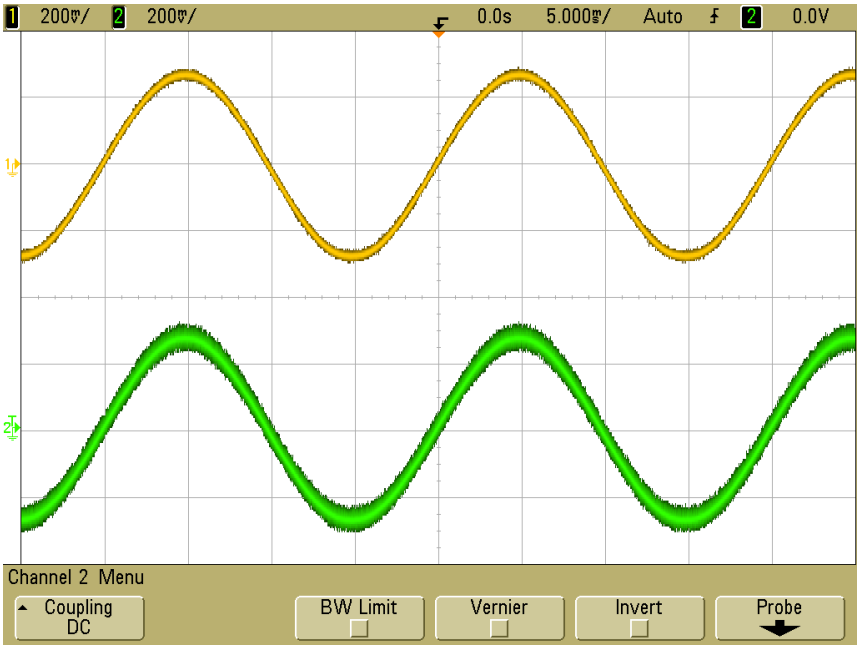


Imagen 4.3.5 – Reducción de ruido en la señal tomada de la protoboard

Aun así la señal sigue teniendo ruido. Este método se complementa con la implementación de filtros paso-bajos, como el filtro antialiasing que se describe en el apartado 4.5 - Filtro antisolapamiento, o el filtro IIR que se describe en el apartado 4.5 - Cálculo de valores eficaces.

4.4. AJUSTE DEL RANGO DINÁMICO.

ADE7759 tiene una función de muestreo de onda que toma muestras de la señal analógica de entrada y las envía al microcontrolador mediante el bus SPI para que este procese los datos. Las muestras obtenidas son palabras de 20 bits, en formato complemento a 2, donde el bit más significativo (bit 19) indica el signo. Una palabra binaria de 19 bits puede representar un número decimal entero entre 0 y 262.144. ADE7759 opera con un valor nominal que es el 63% del fondo de escala, por lo que el valor máximo que contendrá la palabra de 19 bits corresponde al valor decimal 165.151:

$$2^{18} = 262.14$$
$$262.144 \cdot 0{,}63 = 165.150{,}72 \sim 165.151$$

Aplicando la función de transferencia de los ADC que aparece en el datasheet de ADE7759 se puede calcular la tensión de operación nominal en los canales de entrada:

$$\text{Código} = 3.0492 \times \frac{V_{IN}}{V_{REF}} \times 262{,}144 \qquad \textit{Ec (4.4.1)}$$

$$V_{IN} = \frac{V_{REF} \cdot \text{Código}}{3.0492 \cdot 262{,}144} = \frac{2{,}42 \cdot 165.151}{3.0492 \cdot 262{,}144} = 0{,}5\ \text{V}$$

Mediante el ajuste del rango dinámico se elige la resolución de los datos de entrada para alcanzar la sensibilidad requerida

$$\text{Resolución} = \frac{1}{2^{n-1}} \qquad \textit{Ec (4.4.2)}$$

$$\text{Sensibilidad} = \frac{V_{REF}}{(3{,}0493)} \cdot \text{Resolución} \qquad \textit{Ec (4.4.3)}$$

El límite a la hora de elegir el número de bits de la muestra es la sensibilidad que se desee en la medida; por cada bit que se elimina la sensibilidad se multiplica por dos, ya que para que se produzca un cambio en el LSB de nuestra muestra, antes tienen que haber cambiado los bits que hemos desechado. Mediante la función de transferencia del ADC se calcula la sensibilidad, considerándola como el incremento que debe haber en la

tensión de la señal de entrada del convertidor para que el valor de la muestra incremente una unidad. Este valor es de $3{,}027\ \mu V$:

$$V_{IN} = \frac{2{,}42 \cdot 1}{3.0492 \cdot 262{,}144} = 3{,}027\ \mu V$$

Por ejemplo, si la señal del canal de tensión de se obtiene mediante un divisor de tensión con una resistencia de 510kΩy 1kΩ, y la salida es la tensión en la resistencia en 1kΩ, la relación entra la señal en la entrada de ADE7759 y la tensión de la línea que se mide es de 511:

$$V_{IN,LINEA} \cdot \frac{1K\Omega}{510K\Omega + 1K\Omega} = V_{IN,ADE}$$

$$V_{IN,LINEA} = 511 \cdot V_{IN,ADE} \qquad \textit{Ec (4.4.4)}$$

$$0{,}01V \text{ en la entrada} \rightarrow 1{,}957 \cdot 10^{-5} V \text{ en ADE7759}$$

Esto significa que para tener una sensibilidad de 0,01 Voltio en el canal de tensión, necesito que en las muestras pueda ver incrementos de al menos $1{,}957 \cdot 10^{-5}$V. En las siguientes gráficas se puede ver la sensibilidad en cada canal según la relación Señal a medir/ Señal en ADE7759 en función del número de bits de la muestra:

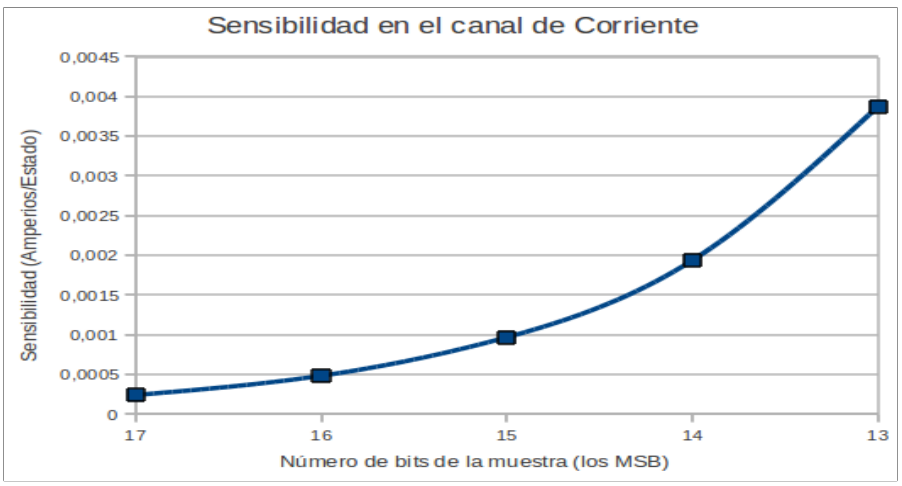


Imagen 4.4.1 – Sensibilidad de la medida de corriente

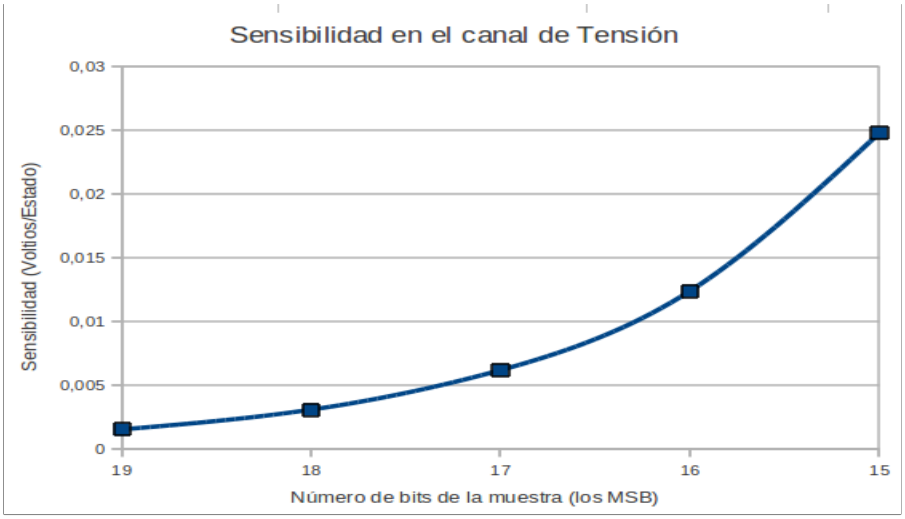


Imagen 4.4.2 – Sensibilidad en la medida de tensión

La siguiente tabla es un resumen de los resultados de los cálculos anteriores. En la primera columna se indican el número de los bits de la muestra; el máximo es 19 porque se elimina el MSB que indica el signo. La segunda columna indica la resolución que se tendría con ese número de bits (ecuación 4.4.2). La tercera columna contiene la sensibilidad de la conversión analógica digital (ecuación 4.4.3), y las columnas cuarta y quinta muestran la sensibilidad para cada magnitud que se quiere medir, considerando la relación sensor/señal entrada a ADE7759. A luz de los datos de la tabla se observa que para obtener una exactitud de 10mA en el canal de corriente, se necesitan muestras de 12 bits como mínimo, y para obtener una exactitud de 10mV en el canal de tensión se necesitan 17 bits. Para la medida de potencia se usan los 19 bits disponibles, y la resolución es de 0,5W por estado aproximadamente:

			1A = 50mV 10A = 0,5V	*1V = 2 mV 255,5V = 0,5V*
Bits	**Resolución**	**Sensibilidad**	**Sensibilidad en Corriente**	**Sensibilidad en Tensión**
19	3,8147E-06	3,02754E-06	6,05508E-05	0,001547072
18	7,62939E-06	6,05508E-06	0,000121102	0,003094143
17	1,52588E-05	1,21102E-05	0,000242203	0,006188287
16	3,05176E-05	2,42203E-05	0,000484406	0,012376573
15	6,10352E-05	4,84406E-05	0,000968812	0,024753147
14	0,00012207	9,68812E-05	0,001937624	0,049506293
13	0,000244141	0,000193762	0,003875248	0,099012587
12	0,000488281	0,000387525	0,007750496	0,198025174
11	0,000976563	0,00077505	0,015500992	0,396050347
10	0,001953125	0,001550099	0,031001984	0,792100694
9	0,00390625	0,003100198	0,062003968	1,584201389
8	0,0078125	0,006200397	0,124007937	3,168402778
7	0,015625	0,012400794	0,248015873	6,336805556

Numero de bits	**Sensibilidad de Potencia (Watios/estado)**
19	0,048732572
18	0,097465143
17	0,194930287
16	0,389860573
15	0,779721147
14	1,559442293
13	3,118884587
12	6,237769174
11	12,47553835

4.5. FILTROS ANTISOLAPAMIENTO

Los filtros antisolapamiento son filtros paso-bajos que se sitúan antes de la entrada analógica de los ADC. Se usan para prevenir el efecto de aliasing. La siguiente imagen ilustra este efecto:

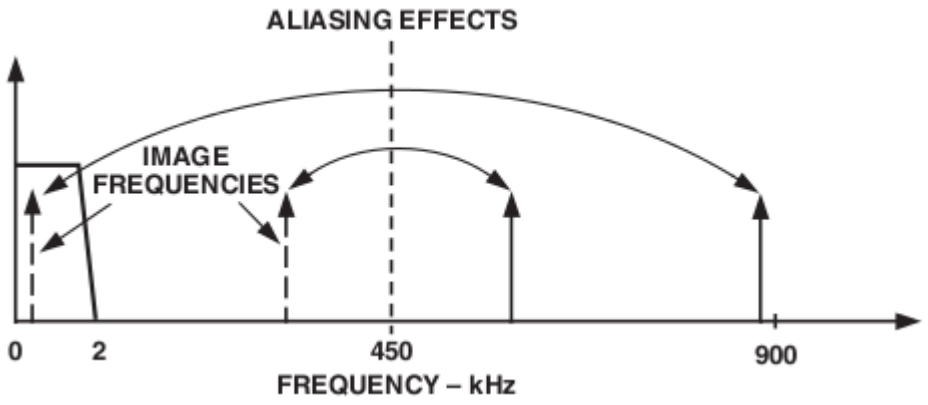


Imagen 4.5.1 – Efecto de Aliasing – Datasheet de ADE7759

Esta imagen muestra como el efecto de solapamiento puede introducir imprecisiones en la medida. Las componentes de frecuencia (flechas mostradas en negro) sobre la mitad de la frecuencia de muestreo (conocida como frecuencia Nyquist) son mostradas o captadas bajo 450 kHz (flechas mostradas con líneas discontinuas), Esto sucederá con todos los ADC sin importar la arquitectura. Solo las frecuencias cercanas a la de muestreo se moverán a la banda de interés de medición, p.e. 0 kHz – 2 kHz. Un filtro LFP (Filtro Paso-bajo) atenuará estas altas frecuencias y prevendrá así la distorsión de la banda de interés. La forma más simple de LPF es un filtro RC con un solo polo y una atenuación de -20dB/dec.

4.6. CÁLCULO DE VALORES EFICACES

Para calcular el valor eficaz mediante las muestras que obtiene ADE7759, es necesario un algoritmo de procesamiento de muestras digitales. Este trabajo se ha basado en el algoritmo recomendado en las notas de aplicación AN-578 que suministra Analog Devices (fabricante de ADE7759).

El valor eficaz de una señal AC es la amplitud que debería tener una señal DC para que ambas produzcan una cantidad de calor equivalente en la misma carga. Definido matemáticamente, el valor RMS de una señal continua V(t) se define como:

$$V_{RMS} = \sqrt{\frac{1}{T} \times \int_0^T V^2(t)} \qquad \textit{Ec (4.5.1)}$$

En la expresión anterior V_{RMS} es el valor eficaz de la señal, y T el periodo de la señal, o el tiempo considerado para una señal no periódica. Si se trabajan con señales muestreadas, este cálculo implica la potenciación de la señal al cuadrado, la realización del promedio y la obtención de la raíz cuadrada para cada muestra entrante.

$$V_{RMS} = \sqrt{\frac{1}{N} \times \sum_1^N V^2(t)} \qquad \textit{Ec (4.5.2)}$$

Para ondas sinusoidales sin distorsión, se puede utiliza el valor de pico (V_p) de la señal para medir el valor eficaz. Este se obtiene multiplicando el valor medio por un factor de forma, característico del tipo de señal. En el caso de la señal sinusoidal, se obtiene:

$$V_{RMS} = \frac{\pi}{2\sqrt{2}} \cdot V_p \qquad \textit{Ec (4.5.3)}$$

Esta solución es adecuada para señales sinusoidales sin distorsión, ya que si cambia la forma de la señal de entrada también lo hará el factor de forma en función del tipo de onda, es decir, la relación Vrms/Vp es diferente, y al cambiar esta se introduce un error significativo en la medida si la señal de entrada no es sinusoidal. El cálculo del valor eficaz que se ha implementado se basa en un método de promediado con un filtro

digital paso-bajo. El tiempo de establecimiento de este filtro determinará el número de muestras necesarias para cometer el menor error posible.

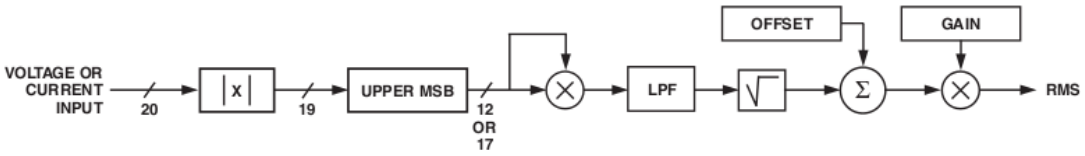


Imagen 4.5.1 – Cadena de procesamiento para cálculo RMS- Notas de aplicación AN-578

La imagen anterior es el diagrama de flujo con las operaciones necesarias del algoritmo para el cálculo de valores eficaces que propone el fabricante de ADE7759 en las notas de aplicación AN-578. En él se muestra los procedimientos que componen el algoritmo de procesamiento de la señal, el cual se divide en dos etapas: el proceso en tiempo real y el post-procesamiento.

- **Proceso en tiempo real**

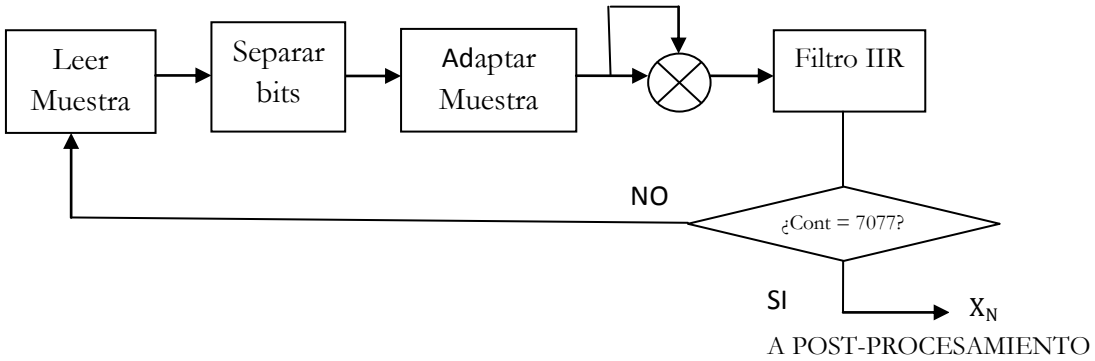
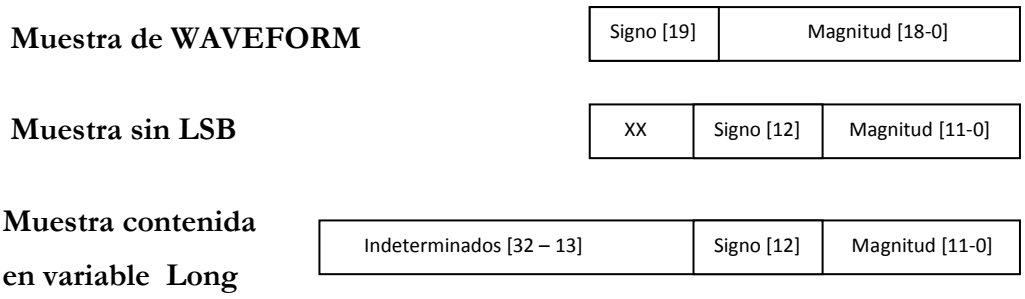


Imagen 4.5.2 – Operaciones del procesamiento en tiempo real

El flanco de bajada de la señal de interrupción $\overline{\text{IRQ}}$ de ADE7759 inicia el proceso en tiempo real, disparando una interrupción externa en el microcontrolador, que mediante la rutina de interrupciones leerá el registro de estado de interrupción de ADE7759 para saber que suceso la ha provocado. Si el motivo de la interrupción es que hay una muestra disponible en el registro de forma de onda WAVEFORM, Arduino leerá el por el puerto SPI el valor de esta muestra, que es una palabra digital de 20 bits, en formato complemento a 2, y la almacenará en una variable de 4 bytes (32 bits).

El siguiente paso consiste en desplazar la variable hacia la derecha un número de bits adecuado, para eliminar los LSB de la muestra. Por ejemplo, si la muestra es de 20 bits y desplazamos 7 bit quedará una palabra de 13 bits, con los 12 bits menos significativos conteniendo el valor de la muestra y el MSB conteniendo el signo.

En el lenguaje de programación de Arduino, las variables almacenan los valores negativos en complemento a dos. Para que la variable contenga realmente el valor de la muestra, debe rellenarse con 'unos' desde el MSB que queda de la muestra hasta el MSB de la variable que lo contiene, que indica el signo (negativo cuando es 1). En el caso anterior, se analizaría el bit 12 de la variable que contiene la muestra, y si este es 1 se rellenaría con '1' los bits que contiene el campo "Indeterminados [32-13]". De esta forma, se adapta el verdadero valor de la muestra, que en un principio era de 20 bits, a 4 bytes para que Arduino operar con él.



El cálculo de la potencia cuadrada del valor de la muestra se puede realizar fácilmente multiplicando el valor por el mismo. Sin embargo, hay que considerar que si tenemos una muestra de n bits, en el caso más desfavorable el resultado puede tener 2n bits. La variable que sigue en capacidad al tipo "Long" es el tipo "Long Long", con una capacidad de 64 bits (8 bytes). Por otro lado, el resultado será siempre positivo, por lo que siempre estaremos trabajando con los MSB de la variable a "Long Long" a 0.

El proceso en tiempo real concluye filtrando el valor de $V^2(n)$ mediante un filtro de respuesta infinita a impulso de primer orden. Los filtros IIR son filtros recursivos porque la salida del filtro depende de valores pasados de sí misma. También es llamado filtro promediador porque promedia las muestras de la entrada y por lo tanto suprime

variaciones rápidas. La ecuación de recurrencia general de este tipo de filtro es la siguiente:

$$y[n] = b_0 \cdot x[n] + a_1 \cdot y[n-1] \qquad \textit{Ec (4.5.4)}$$

La función de transferencia del filtro es:

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{b_0}{1 - a_1 \cdot z^{-1}} \qquad \textit{Ec (4.5.5)}$$

La respuesta en frecuencia de este filtro se obtiene evaluando $H(Z)$ en $Z = e^{-j2\pi \frac{f}{f_{muestreo}}}$ El módulo de la respuesta en frecuencia es:

$$H(f) = \sqrt{\frac{{b_0}^2}{1 + {a_1}^2 - 2 \cdot a_1 \cdot \cos(2\pi \cdot \frac{f}{f_{muestreo}})}} \qquad \textit{Ec (4.5.6)}$$

Este filtro tiene un cero en $z = 0$ y un polo $z = a_1$. La condición de estabilidad del filtro es que todos los polos y los ceros estén dentro de la circunferencia unidad, por lo que se tiene que cumplir que $0 < a_1 < 1$. Por otro lado, para que se cumpla la restricción de ganancia unitaria de un filtro paso-bajo, se tiene que cumplir la siguiente relación entre los coeficientes:

$$b_0 + a_1 = 1 \qquad \textit{Ec (4.5.7)}$$

Para incrementar la velocidad de operación del filtro, se diseña de manera que los coeficientes sean una potencia de 2, ya que la multiplicación del valor de una muestra por 2^P se realiza desplazando el contenido del registro que contiene el valor hacia la izquierda P bits. Si la operación es de división, se desplazan hacia la derecha. La siguiente tabla muestra el valor de la frecuencia de corte del filtro (en Hertzios) para la Frecuencia de muestreo elegida, y operando P bits. Para una frecuencia de corte de 0,54 Hz se necesitan 10 bits, con una frecuencia de muestreo de 3,5kHz aproximadamente. Por lo tanto, si se elige $b_0 = 2^{-P}$ entonces $a_1 = 1 - 2^{-P}$. La frecuencia de corte será aquella en la que la ganancia sea -3dB:

$$H(z) = 10^{\frac{-3}{20}} = 0{,}708$$

$$f_{corte} = \cos^{-1}\left(\frac{1+{a_1}^2-\frac{{b_0}^2}{0{,}708^2}}{2\cdot a_1} - 2\cdot a_1\right)\cdot\frac{f_{muestreo}}{2\cdot\pi} \qquad \textit{Ec (4.5.8)}$$

Frecuencia Oscilador (Hz)	**Frecuencia Muestreo 1 (Hz)**	**Frecuencia Muestreo 2 (Hz)**	**Frecuencia Muestreo 3 (Hz)**	**Frecuencia Muestreo 4 (Hz)**
3579545	3495,649414	6991,298828	13982,59766	27965,19531
Número de bits	**Frecuencia de corte (Hz)**	**Frecuencia de corte (Hz)**	**Frecuencia de corte (Hz)**	**Frecuencia de corte (Hz)**
1	401,0958579	802,1917158	1604,383432	3208,766863
2	160,7824061	321,5648123	643,1296246	1286,259249
3	74,22409279	148,4481856	296,8963712	593,7927423
4	35,83325254	71,66650508	143,3330102	286,6660203
5	17,62297001	35,24594003	70,49188005	140,9837601
6	8,740997311	17,48199462	34,96398924	69,92797849
7	4,353224294	8,706448587	17,41289717	34,82579435
8	2,172335926	4,344671851	8,689343703	17,37868741
9	1,085104138	2,170208277	4,340416554	8,680833107
10	0,542286763	1,084573526	2,169147052	4,338294103
11	0,271077136	0,542154272	1,084308543	2,168617086
12	0,135522016	0,271044033	0,542088066	1,084176132
13	0,067756871	0,135513743	0,271027486	0,542054972
14	0,033877402	0,067754804	0,135509607	0,271019215

En la anterior se puede ver el valor de la frecuencia de corte en función de la frecuencia de muestreo y del número de bits que se desplazan P (Ecuación 4.5.8). Este filtro sirve, además de para promediar las muestras, para disminuir el efecto del rizado de frecuencia. Para ello la frecuencia de corte del filtro debe quedar lo más cerca posible

a 0. De este modo se atenuarán las frecuencias mayores de 100Hz mas de 45 dB, lo que significa que la frecuencia de rizado representaría el 0,56% de la señal DC se salida. . La ecuación de recurrencia del filtro aplicando los coeficientes es:

$$y[n] = 2^{-P} \cdot x[n] + (1 - 2^{-P}) \cdot y[n-1]$$

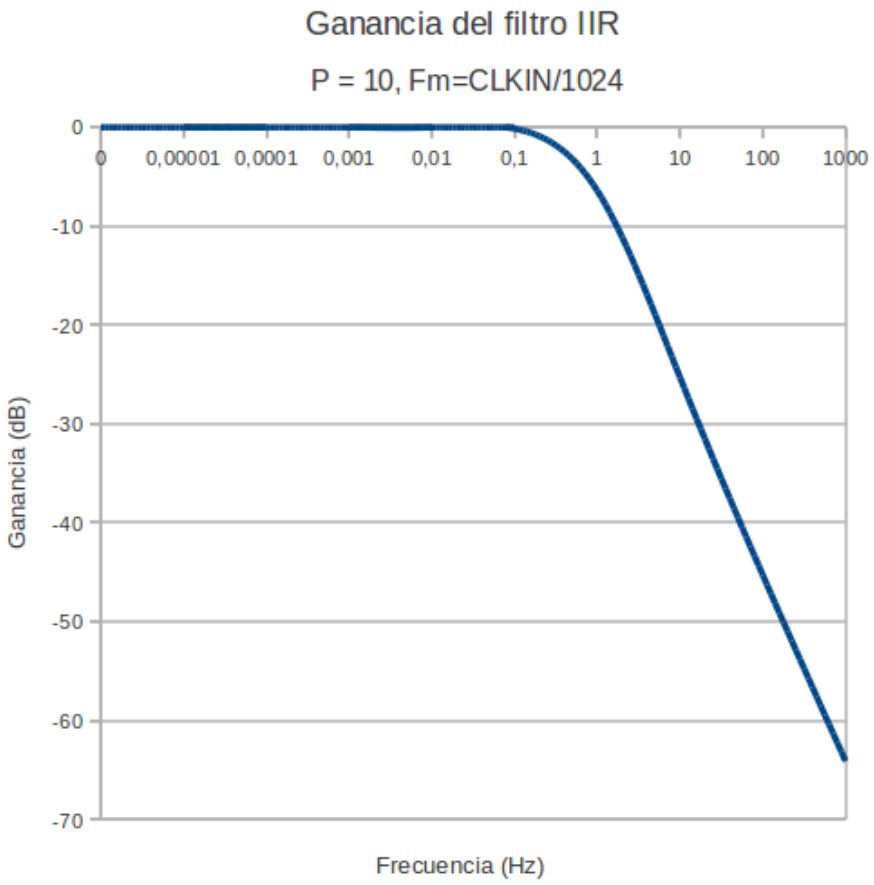


Imagen 4.5.3 - Función de transferencia del filtro IIR

El filtro se implementa con tres instrucciones, como se indica en el siguiente diagrama de flujo:

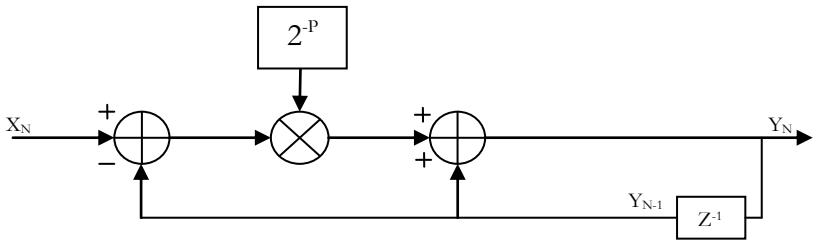


Imagen 4.5.4 – Diagrama de flujo del filtro IIR

En el caso de querer aproximar más a 0 la frecuencia de corte, habría que tomar un valor de P más elevado, y se hace necesario tomar más muestras para una medida precisa. Esto significa que se tarda más tiempo en el proceso de filtrado. La ecuación que representa el error en el filtro IIR es la siguiente:

$$\text{Error} = \frac{1}{(1-2^{-P})^n} \cdot 100 \qquad \textbf{\textit{Ec (4.5.9)}}$$

Aplicando esta ecuación se ha realizado la siguiente tabla. En ella se refleja cuantas muestras son necesarias para obtener un error igual al 0,1%. Cuantas más muestras se procesen, menor será el error. También aparece el tiempo que se tarda en procesar ese número de muestras para cada frecuencia de muestreo (F1 la más baja y F4 la más alta):

P	N. Muestras	Tiempo F1	Tiempo F2	Tiempo F3	Tiempo F4
8	1771,83698	0,506869189	0,253434595	0,126717297	0,063358649
9	3540,22346	1,012751291	0,506375646	0,253187823	0,126593911
10	7076,99472	2,024515014	1,012257507	0,506128753	0,253064377
11	14150,5364	4,048042218	2,024021109	1,012010554	0,506005277
12	28297,6194	8,095096507	4,047548253	2,023774126	1,011887063
13	56591,7851	16,18920502	8,094602512	4,047301255	2,023650628
14	113180,116	32,37742203	16,18871101	8,094355504	4,047177754

Para obtener un error menor del 0,1% en el promedio, con un valor P = 10, a la frecuencia de muestreo más baja, se necesitaran 2,025 segundos, que es el tiempo que se tarda en tomar 7077 muestras

- **Post-Procesamiento**

El post-procesamiento está compuesto por las operaciones que se realizan cuando se termina el muestro para obtener el valor de la magnitud a medir. El diagrama de flujo de las operaciones a realizar es el siguiente

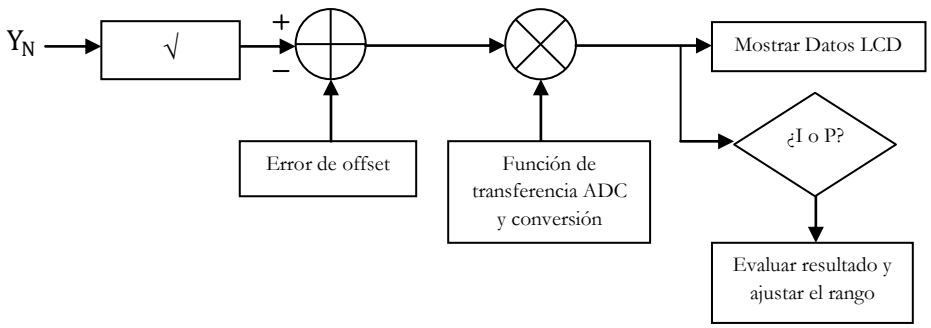


Imagen 4.5.5 – Diagrama de flujo del post-procesamiento

La primera operación que se realiza es el cálculo de la raíz cuadrada. En el peor de los casos trabajaremos con muestras de 19 bits, por lo que el máximo número de bits en la entrada del cálculo de la raíz cuadrada será 38 bits, y en la salida 19 bits. El entorno de Arduino tiene la función **sqrt()** para calcular la raíz cuadrada de una variable.

Una vez que se ha realizado la raíz cuadrada se tiene el valor eficaz de la señal de entrada codificado mediante la función de transferencia del ADC. Este es el momento de realizar la corrección de offset (que se ha calculado previamente en un proceso de calibración) y multiplicar el código obtenido por el coeficiente de conversión. El resultado se almacenará en una variable real "float" y finalmente se envía el resultado al display LCD para mostrar los datos por pantalla. En caso de implementar una selección de autorango, esta se debe hacer en el post-procesamiento. Esta operación consiste en una selección con histéresis que modifica el fondo de escala.

4.7. CÁLCULO DE POTENCIA ACTIVA Y ENERGÍA CONSUMIDA

La potencia de la energía eléctrica se define como la velocidad del flujo de energía desde la fuente a la carga, que se obtiene mediante el producto de las ondas de tensión y corriente. La onda resultante se denomina señal de potencia instantánea, y es igual a la velocidad de flujo de energía en cada instante de tiempo. La unidad de potencia es el

Watt o Julios/segundo. La ecuación 4.6.3 nos da una expresión para la señal de potencia instantánea en un sistema AC

$$v(t) = \sqrt{2} \cdot V_{ef} \cdot \sin(\omega t) \qquad \textit{Ec (4.7.1)}$$

$$i(t) = \sqrt{2} \cdot I_{ef} \cdot \sin(\omega t - \varphi) \qquad \textit{Ec (4.7.2)}$$

$$p(t) = v(t) \times i(t) = 2 \cdot V_{ef} \cdot I_{ef} \cdot \sin(\omega t) \cdot \sin(\omega t - \varphi)$$

$$p(t) = v(t) \times i(t) = V_{ef} \cdot I_{ef} \cdot (\cos(\varphi) - \cos(2\omega t - \varphi))$$

$$\underbrace{p(t) = v(t) \times i(t)}_{\text{Potencia Instantanea}} = \underbrace{V_{ef} \cdot I_{ef} \cdot \cos(\varphi)}_{\text{Potencia Activa}} - \underbrace{V_{ef} \cdot I_{ef} \cdot \cos(2\omega t - \varphi)}_{\text{Potencia Fluctuante}} \qquad \textit{Ec (4.7.3)}$$

ADE7759 genera la señal de potencia instantánea p(t) multiplicando las señales de corriente y tensión. A continuación se extrae la componente DC de esta señal de potencia instantánea mediante el bloque LPF2 (filtro paso-bajo) para obtener la información de potencia activa. Este proceso se ilustra a continuación:

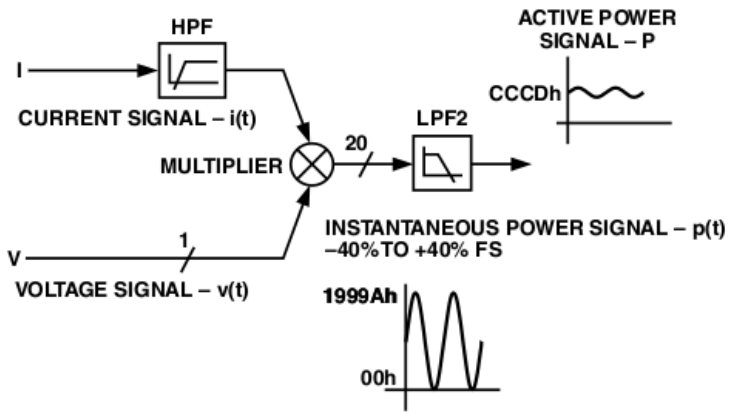


Imagen 4.6.1 – Procesamiento de la señal de potencia activa

El rango del código máximo (hexadecimal) de salida para la señal de potencia activa (LPF2) cuando el integrador digital esta deshabilitado es CCCDh. El rango de salida también modificarse mediante el registro de ganancia de potencia o cambiando la ganancia de los PGA de entrada. El filtro LPF2 no tiene un comportamiento ideal, por lo que la señal de potencia activa tendrá un rizado a consecuencia de la potencia

fluctuante. Este rizado es sinusoidal y tiene una frecuencia igual al doble de la frecuencia de la línea. Las muestras de potencia activa se envían al registro de forma de onda WAVEFORM para ser leídas en el modo de muestreo, o para sumarlas al registro de acumulación de energía.

La potencia activa es igual a la componente DC de la señal de potencia instantánea p(t) en la ecuación 4.6.3 El procedimiento de operación con las muestras es idéntico al descrito en el apartado 4.3 - Método para el cálculo de valores RMS, con la diferencia que en este caso las muestras no están elevadas al cuadrado, y además el resultado puede ser negativo. Al igual que en el caso anterior, el cálculo está formado por dos etapas, el procesamiento en tiempo real y el post-procesamiento. El siguiente diagrama muestra las operaciones que se realizan durante el procesamiento en tiempo real:

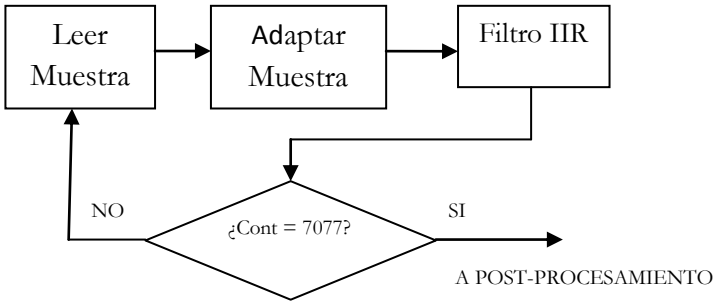


Imagen 4.6.2 – Procesamiento en tiempo real de la señal de potencia activa

El procesamiento en tiempo real consiste en tres operaciones: leer la muestra cuando esté disponible, adaptar los 24 bits de la muestra a la variable de 64 bits con la que se trabaja, y filtrarla mediante el filtro IIR para eliminar el rizado. El número de muestras necesarias se calcula en función del tiempo de establecimiento del filtro:

$$\text{Error} = \frac{1}{(1-2^{-P})^n} \cdot 100 \qquad \textbf{\textit{Ec (4.7.4)}}$$

Para un error menor del 0,1% y a la velocidad de muestreo CLKIN/1024, usando un filtro con P = 10 se necesitan 7.077 muestras, que representan un tiempo de 2,03 segundos. Una vez se tiene el código correspondiente a la potencia activa se realiza el post-procesamiento, que consiste en multiplicar el código obtenido por el coeficiente de

conversión adecuado para mostrar el valor real de la potencia activa. En esta ocasión, el valor promediado será el doble puesto que la señal de potencia instantánea tiene el doble de frecuencia que la línea. Por ello, en vez de calcular la raíz cuadrada se divide entre dos.

La potencia se define como la velocidad de flujo de energía. Esta relación se puede expresar matemáticamente como la variación de energía respecto al tiempo, e inversamente se puede obtener la energía mediante la integral de la potencia:

$$P = \frac{dE}{dt} \qquad \textit{Ec (4.7.5)}$$

$$E = \int p(t) \cdot dt \qquad \textit{Ec (4.7.6)}$$

$$E(t) = \int_0^{nT} VIdt - \left(\frac{VI}{4\pi f_l (1+2f_l/8.9Hz)}\right) \int_0^{nT} \cos(2\omega t)\, dt \qquad \textit{Ec (4.7.7)}$$

Donde *n* es un número entero y T es el periodo del ciclo de la línea. El procedimiento de acumulación de energía consiste en sumar el valor de la muestra instantánea en un registro durante un tiempo determinado. Como la componente sinusoidal se integra durante un número entero de ciclos, este valor es siempre cero:

$$E(t) = \int_0^{nT} VIdt + 0 = VInT \qquad \textit{Ec (4.7.8)}$$

El inicio de la acumulación está sincronizado con el cruce por cero de la señal del canal 2, de modo que la acumulación puede realizarse durante el número de periodos que se configure. Realizando el promedio de potencia instantánea sobre una integral definida entre un número entero de periodos (nT) se obtiene el valor de la potencia activa:

$$P_{Activa} = \frac{1}{nT} \int_0^{nT} p(t)dt = V_{ef} \cdot I_{ef} \cdot \cos(\varphi) \qquad \textit{Ec (4.7.9)}$$

Donde T es el periodo de la señal de potencia instantÁnea, y n es el número de ciclos. Este parámetro se configura mediante el registro LINECYC (14h). Este registro puede contener un valor entre 0 y 16.383 (decimal). El registro LINECYC se

decrementará una unidad en cada paso por cero, por lo tanto, un LSB de LINECYC supone un semiperiodo:

$$n \cdot T = \text{LINECYC} \cdot \frac{1}{2} \cdot f_{linea} \qquad \textit{Ec (4.7.10)}$$

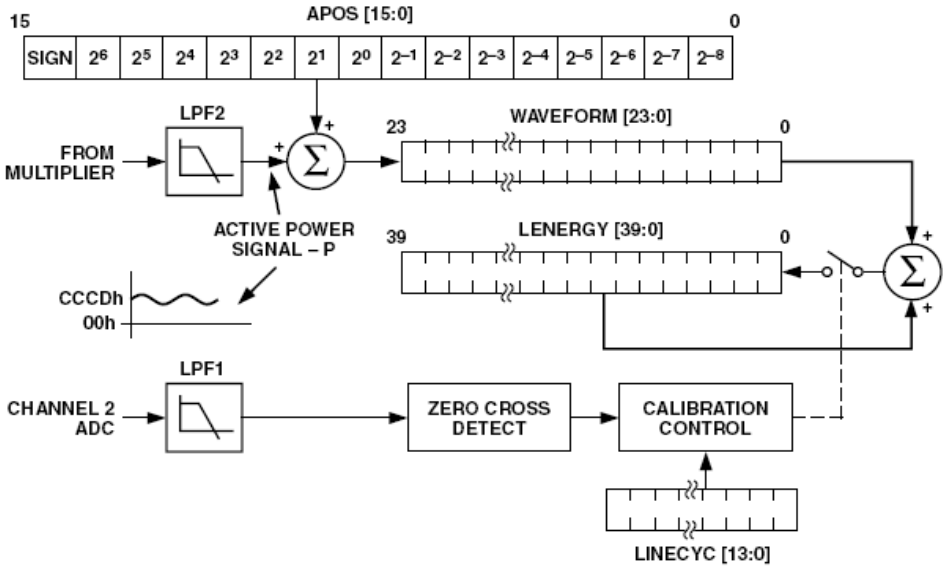


Imagen 4.6.3 – Procesamiento para el cálculo de energía consumida

Cuando LINECYC llegue a cero, la marca CYCEND en el registro de estado de interrupción (bit 2) pasará a nivel alto (en el caso que el bit CYCEND del registro de habilitación de interrupción esté configurado a nivel alto) y la salida de solicitud de interrupción $\overline{\text{IRQ}}$ disparará la rutina de interrupción. En consecuencia la línea $\overline{\text{IRQ}}$ puede usarse para completar la señal de acumulación de energía del ciclo de la línea.

Cuando el bit 7 CYCMODE en el registro de modo se configura a nivel lógico 1, la energía se va acumulando durante un número entero de semiperiodos. Al ser la frecuencia de línea constante (50Hz) el número de semiperiodos de integración también lo será. El tiempo de integración total se obtiene como:

$$\frac{1}{2 \times 50\ \text{Hz}} \times \text{numero de semiperiodos} \qquad \textit{Ec (4.6.10)}$$

Para 255 semiperiodos obtendremos un tiempo total de integración de 2,55 segundos. Esto significaría que el registro de energía se cargaría 2,55/1,1175 μs veces. El valor medio de salida del LPF2, en términos de contenido de varios registros del ADE7759, CLKIN y la frecuencia de la línea (f_l) es:

$$\text{Valor Medio (LPF2)} = \frac{\text{LENERGY [39:0]} \times 4 \times f_l}{\text{LINECYC [13:0]} \times \text{CLKIN}} \qquad \textbf{\textit{Ec (4.6.10)}}$$

4.8. ERRORES EN LA MEDIDA

Como consecuencia de interferencias entre los canales en el PCB o en el propio CI, puede aparecer desviaciones en la medida. También es necesario considerar todos los bloques por los que pasa la señal para ser procesada, ya que estos introducirán desfases y atenuaciones de las señales que procesa La calibración de descompensaciones permitirá aproximar la lectura a cero, o lo más cercano posible, cuando no haya señal de entrada.

- **ERROR DE CUANTIZACIÓN**

La exactitud en el proceso de digitalización de la muestra analógica está determinada por el error de indecisión digital. Esto significa que el valor obtenido de la muestra puede estar una unidad por encima o por debajo de su valor real, por lo tanto la exactitud dependerá de la sensibilidad y del valor que tengamos que medir:

$$\text{Error de indecisión} = \frac{\text{Sensibilidad}}{\text{Valor a medir}} \cdot 100 \qquad \textit{Ec (4.8.1)}$$

Por ejemplo, para medir un valor de 0,5V, si la muestra es de 19 bits, incrementa un bit cada 3,027μV. Si eliminamos los dos últimos bits, el incremento tendrá que ser de 12,108μV, y si se eliminan los 8 últimos bits la sensibilidad pasa a ser 774,912 μV/estado. Por lo tanto, cuantos más bits tenga la muestra, menor será el error por indecisión y en consecuencia la digitalización será más exacta.

El mínimo error en el proceso de cuantización se obtendrá cuando el valor de la entrada al convertidor sea máxima. Por ejemplo: tenemos 165.151 estados con un fondo de escala de 0,5V, y la sensibilidad de la conversión analógica digital es de 3,027μV por estado. Si medimos una señal de 0,5V, el error relativo que se comete por la imprecisión digital de ±1 estado, significa un error en la medida de un $6{,}054 \cdot 10^{-6}$ por unidad, en cambio para medir una señal de 0,25 V se tendría un error relativo el doble del anterior, de $12{,}108 \cdot 10^{-6}$ por unidad.

$$\frac{3{,}027\mu\text{V}}{0{,}5\text{V}} = 6{,}054 \cdot 10^{-6}$$

$$\frac{3{,}027\mu V}{0{,}25V} = 12{,}108 \cdot 10^{-6}$$

Para cometer el menor error posible en la digitalización de la señal de entrada hay que intentar que el valor de esta en los convertidores ADC esté lo más cercana posible al valor nominal de operación de los convertidores. ADE7759 incluye dos amplificadores diferenciales de instrumentación con ganancia programable con los que se puede amplificar la señal de entrada antes de pasar por los convertidores. Volviendo al ejemplo anterior, si se multiplica por 2 la señal de 0,25V, el convertidor verá en la entrada una señal de 0,5V y el error de imprecisión digital será igual que en el primer caso. Otra forma de obtener el mismo resultado es cambiando el valor de fondo de escala modificando la tensión de referencia de los convertidores.

El error por indecisión digital en el cálculo RMS será mayor cuando el valor de tensión en la entrada sea bajo. La siguiente imagen muestra en azul el error de indecisión digital en la medida del canal de tensión cuando el fondo de escala es 0,5V, y en rojo se muestra el error que se comete en la misma situación pero con autoselección de rango. Esto significa que cuando la señal llega a 0,25V, el rango se cambiará automáticamente:

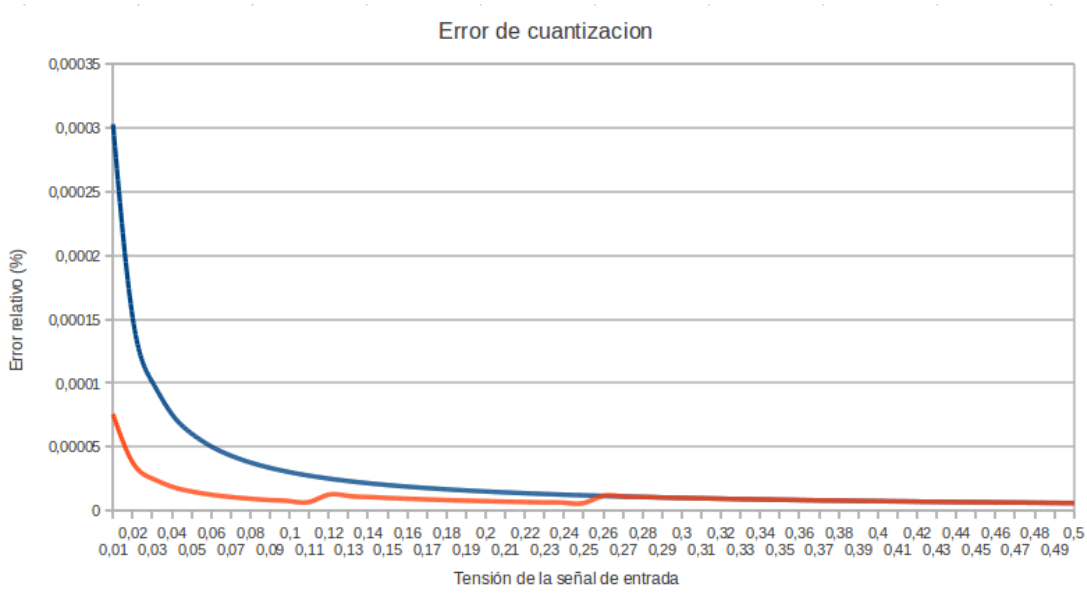


Imagen 4.8.1 - Error de cuantización para Ganancia = 1 (Azul) y con autorrango (roja)

- **ERROR DE OFFSET**

El nivel DC de los canales del ADC introduce un error en la medida del sistema. Por ejemplo, tras la operación de potenciación, una onda sinusoidal con un offset pequeño como $V(t) = V_{OS} + V_{COS}(\omega t)$ produce:

$$V_{OS}^2 + \frac{V^2}{2} + 2 \cdot V_{OS}^2 \cdot V \cdot \cos(\omega t) + \frac{V^2}{2} \cdot \cos(2\omega t) \qquad \textbf{\textit{Ec (4.8.2)}}$$

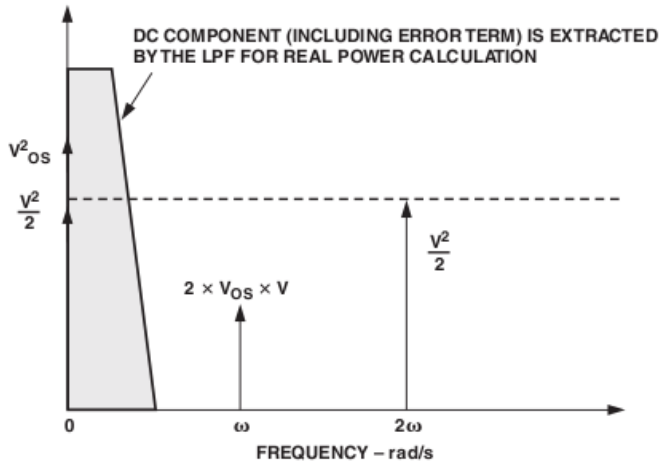


Imagen 4.8.2 – Error de offset en el calculo RMS- Notas de aplicación AN-578

La siguiente imagen muestra el efecto de offset en el cálculo de la potencia activa; un error de offset en el canal 1 y canal 2 contribuiría a una componente continua DC después de la multiplicación.

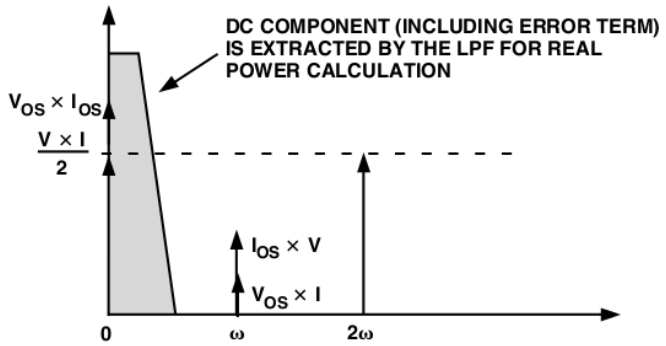


Imagen 4.8.3 – Error de offset en el cálculo de potencia activa – Datasheet ADE7759

El problema del offset en el cálculo de potencia activa puede solucionarse habilitando el integrador HPF1 del canal 1; eliminando el offset desde al menos un canal, no se genera ninguna componente de error en DC por la multiplicación. Sin embargo, en el cálculo RMS no es posible aplicar esta solución.

- **ERRORES DE FASE Y GANANCIA**

La respuesta en fase de los distintos bloques de procesamiento de la señal, puede introducir errores significativos si ambos canales no están en fase. Una posible fuente de errores de fase externos es el filtro anti-solapamiento del canal 1 y el canal 2, junto con el desplazamiento en fase que introduce el sensor de corriente. El desacoplo puede ocurrir fácilmente debido bajas tolerancias de los componentes en el LPF de entrada (Filtro antisolapamiento). En la siguiente imagen se puede ver la respuesta en fase para un LPF como el diseñado a 50 Hz para R = 1 kΩ ±10%, C = 33 nF ±10%. Un desplazamiento de fase de 0.2º puede provocar un error de medida de potencia activa de 0.6% para bajos factores de potencia. El diseño usa resistencias con una tolerancia del 1% y condensadores con tolerancia de 10% para los reducir posibles problemas debidos a desacoplos. Alternativamente, el codo de frecuencia del filtro moverse a 10 kHz – 15 Hz. Sin embargo, el codo de frecuencia no debe hacerse muy alto, ya que esto permitiría bastantes componentes de alta frecuencia, lo que se solaparía causando problemas en ambientes ruidosos:

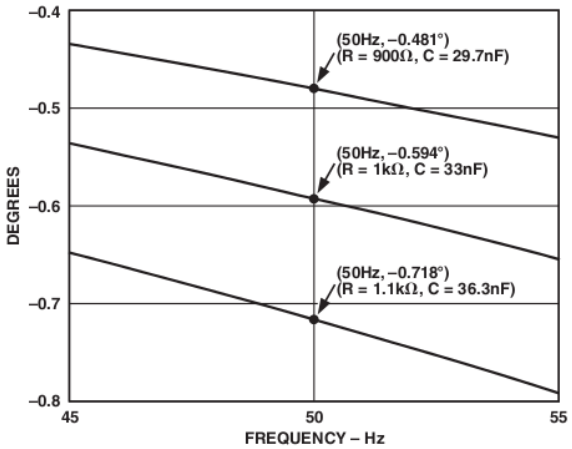


Imagen 4.8.4 – Desfase que introduce los deslizamientos en los valores de los componentes

Por otro lado, en el interior de ADE7759 el error de fase entre el canal 1 y el canal 2 es cero en un rango de frecuencias que va desde DC hasta 3.5 kHz cuando HPF1 no está habilitado, Cuando HPF1 está habilitado, el canal 1 tiene una respuesta en fase mostrada a continuación.

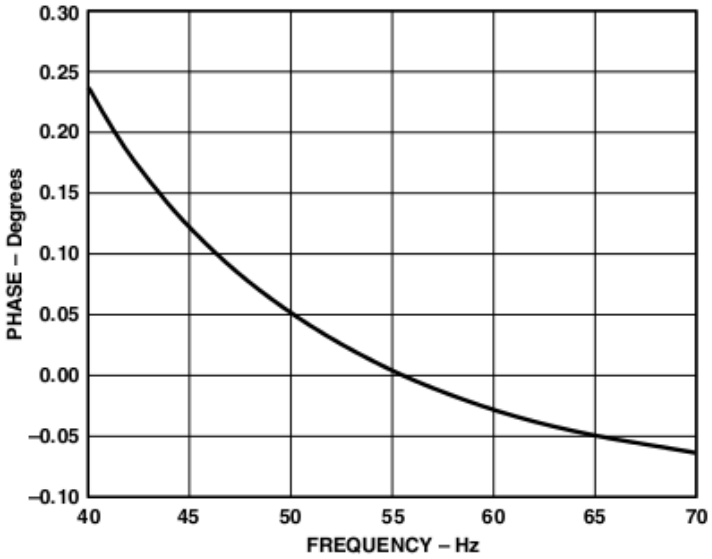


Imagen 4.8.5 – Respuesta en fase de HPF1

Como puede observarse en la grafica, el filtro desplaza la señal de corriente 0,05º a 50Hz. Por otro lado los transductores podrían tener errores de fase inherentes, siendo habitual que un sensor CT de corriente introduzca un error de fase de 0,1º a 0,8º. Este error puede variar de un dispositivo a otro y deben ser corregidos para realizar cálculos de potencia precisos. Los errores asociados con desacoplos de fase son particularmente notables con bajos factores de potencia. ADE7759 proporciona un medio para calibrar digitalmente estos pequeños errores de fase mediante la introducción de un pequeño tiempo de retraso o tiempo de avance en la cadena de procesamiento de la señal de entrada de tensión para compensar pequeños errores de fase. El fabricante recomienda usar esta técnica para corregir errores de fase en el rango de 0.1° a 0.5°, por lo que el desfase introducido por el sensor puede ser una limitación a la hora de diseñarlo.

Las señales de tensión y de potencia instantánea son filtradas internamente por filtros paso-bajo (LPF1 y LPF2 respectivamente) para atenuar componentes de altas frecuencias. Estos filtros introducen un desplazamiento en la señal, además de una

atenuación en la componente con la frecuencia a medir. A continuación se muestra una gráfica con la respuesta del filtro LPF1. Este filtro tiene la frecuencia de corte en 156Hz, y presenta una atenuación de 0.6dB para una señal de 50Hz, lo que significa que si el valor eficaz de la señal es de 0,5 V, el medidor mostraría en la salida 0,475Voltios:

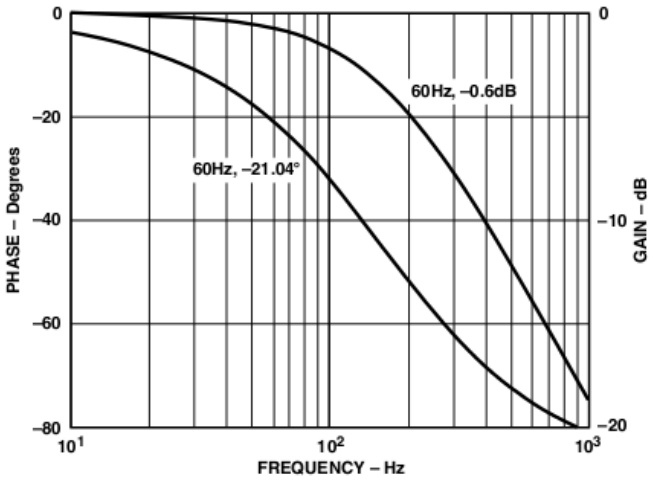


Imagen 4.8.6 – Característica de LPF1- Datasheet de ADE7759

4.9. CALIBRACIÓN DEL MEDIDOR

- **Calibración de tensión y corriente eficaz**

El proceso de calibración del medidor consiste en aplicar una entrada y observar el código que se tiene a la salida. Si se realiza la calibración con un generador estable, es decir, una corriente con valor estable y una tensión también estable, solo es necesario calibrar el medidor en un puntos; si por el contrario no se dispone del material necesario se pueden realizar múltiples medidas y calcular la recta de regresión.

Para un valor de entrada estable, por ejemplo 230 Voltios, se debe de tener un código en la salida igual a 387:

$$\text{Código} = \frac{\text{Entrada}}{\text{Sensibilidad}} = \frac{230 \text{ Voltios}}{0{,}00618 \text{ Voltios /Estado}} = 37216 \quad \textit{Ec (4.9.1)}$$

La lectura realizada debe ser 37216. Hay que considerar las atenuaciones de los filtros internos de ADE7759; HPF1 en el cálculo de corriente y LPF1 en el cálculo de tensión, y el error cometido en el proceso de medida. El error de offset será la diferencia entre el código obtenido y el código esperado a la salida; si el código en la salida es menor del esperado, la tensión de offset es negativa, y si el código es mayor la tensión de offset positiva.

El cálculo de potencia activa no debe verse afectado por error de offset ya que tiene un filtro paso-alto que los elimina. Aun así si hubiese alguno el método de cálculo es el mismo que para el error de offset de tensión.

- **Calibración de la potencia activa**

La calibración de potencia activa en el ADE7759 consiste en determinar el valor medio de la señal de salida de LPF2 para un valor de entrada conocido. La calibración se puede realizar mediante el modo de acumulación de energía o mediante la extracción de la componente continua de la señal de potencia instantánea. Es importante que la carga que se emplee sea resistiva, dicho de otro modo, que la corriente y la tensión en la carga

estén en fase. El procedimiento es igual que el anterior; conectar una carga que consuma una potencia conocida y operar el código de salida.

Para realizar la conversión del código se aplica una regla de tres; si cuando las entradas son las máximas posibles con un factor de potencia unitario tenemos 2555 Watios/segundo (10 Amperios en el canal de corriente y 255,5 Voltios en el canal de tensión) y el código de salida es 52.429, la sensibilidad es de 0,04873 Wat/seg:

$$\frac{\text{Potencia}_{\max}}{\text{Codigo}_{\max}} = \frac{2555}{52.429} = 0.04873 \ {}^{\text{Wat}}/_{\text{seg}} \qquad \textit{Ec (4.9.2)}$$

Por lo tanto, ante una potencia de entrada estable y conocida se puede calcular el resultado esperado:

$$\text{Potencia} = \text{Código} \cdot 0{,}04873 \qquad \textit{Ec (4.9.3)}$$

Al realizar la primera medida se observará que la lectura no coincide con el valor real. Esto se debe al desplazamiento de fase que se produce en los distintos bloques de filtrado de la señal. Mediante el registro de calibración de fase PHCAL, el tiempo de retardo en el camino de la señal del canal 2 puede variarse desde –110 μs hasta +103 μs (CLKIN = 3.579545 MHz). Un LSB es equivalente a 1.12 μs de retraso o avance. Para una frecuencia de línea de 50 Hz, se obtiene una resolución de fase de 0.02° (360° × 1.12 μs × 50 Hz). Para calcular el desfase se aplica la siguiente ecuación:

$$\varphi = \arccos\left(\frac{Codigo\ esperado}{Codigo\ obtenido}\right) \qquad \textit{Ec (4.9.4)}$$

$$Valor\ PHCAL = \frac{\varphi}{0.02} \qquad \textit{Ec (4.9.5)}$$

5. RESULTADOS

5.1. IMPLEMENTACION DE DISPLAY LCD EN ARDUINO

En la actualidad existen multitud de alternativa para la visualización de los datos obtenidos por el sistema diseñado. Se están desarrollando componentes en los últimos años como las pantallas táctiles, que además de presentar los datos de interés, ofrecen una interacción directa entre el usuario y el dispositivo. Esta alternativa facilita el uso del sistema, aunque es una opción de alto coste económico:

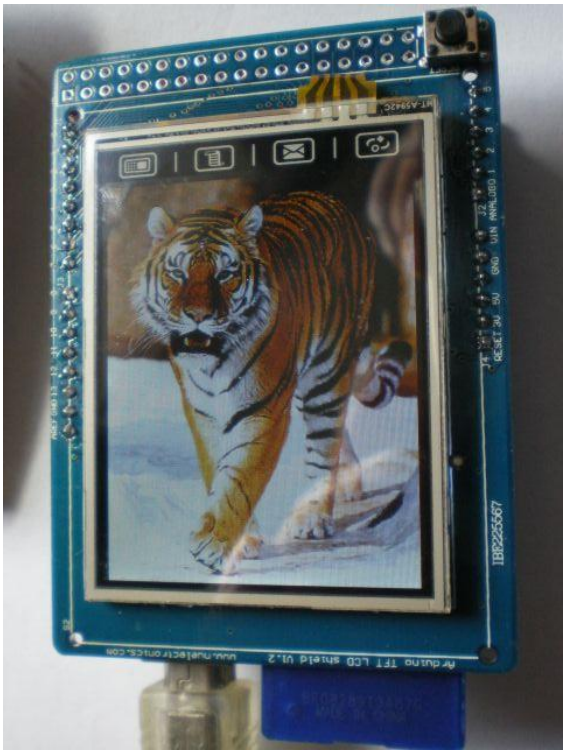


Imagen 5.1.1 - Pantalla tactil TFT

En esta foto se puede observar una pantalla TFT soldada en una PCB para poder conectarlo directamente sobre una placa Arduino modelo Duemilanove. Otra opción más económica son los displays LCD 16x2. Siempre que se busque optimizar el coste del dispositivo y no se requiera pintar gráficas, es más interesante utilizar pulsadores para implementar el control del usuario y este display para visualizar los datos. Estos dispositivos permiten visualizar mensajes alfanuméricos.

Imagen 5.1.2 - Display LCD 16x2. Imagen obtenida de Ebay

Para este trabajo se ha utilizado un display LCD 16x2. En el anexo II; Pantalla de cristal líquido (LCD**)** hay información más detallada sobre el funcionamiento del LCD

- **Esquema eléctrico del circuito**

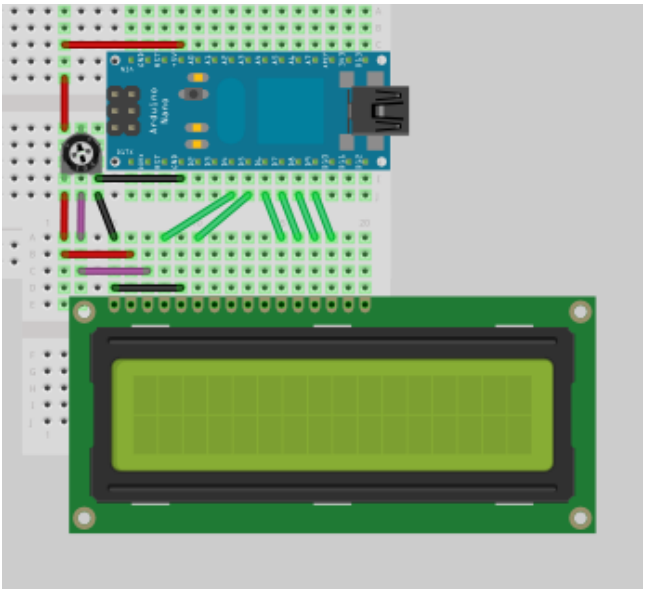


Imagen 5.1.3 - Vista del circuito implementado en una placa de prototipado con Arduino Nano

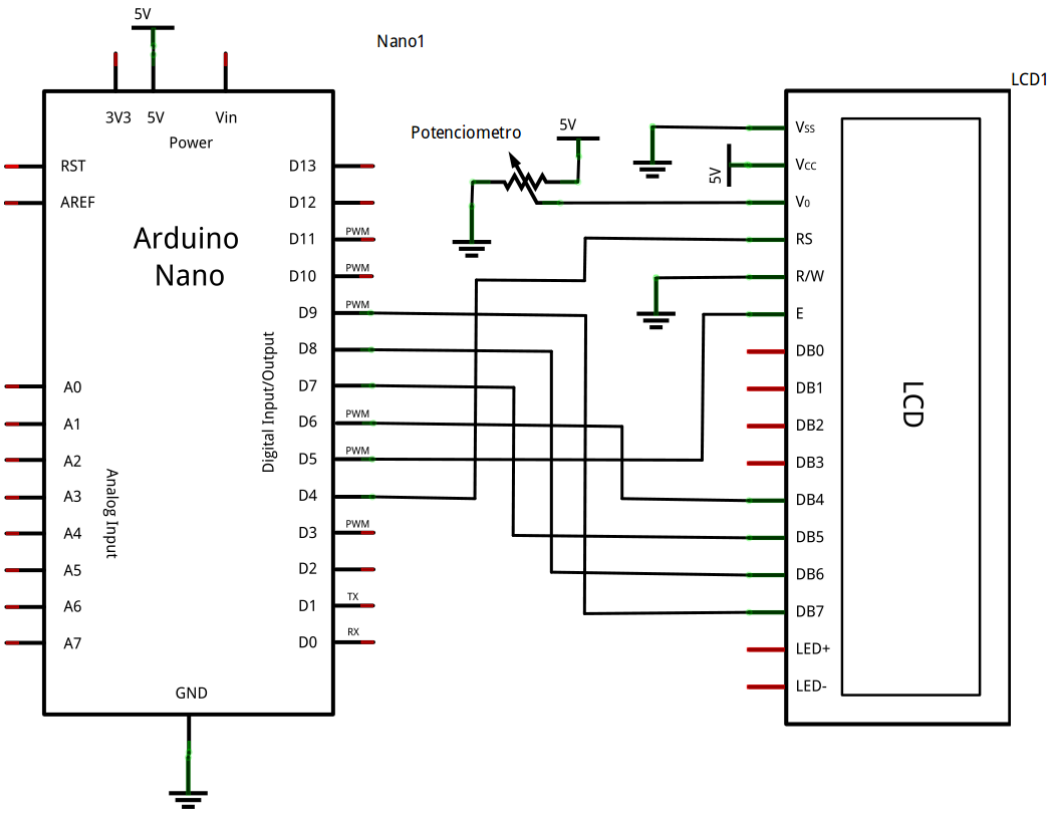


Imagen 5.1.4 - Esquema para la implementación del display LCD

- **Descripción del funcionamiento**

El display LCD se alimenta directamente desde la placa Arduino conectado a los terninales 5V y GND, y su consumo es aproximadamente 8 mW. Mediante un potenciómetro se ajusta el contraste de la pantalla. El protocolo de comunicación del display LCD consiste en una comunicación en paralelo, con la posibilidad de usar un ancho de 8 bits (DB0-DB7) o de 4 bits (DB4-DB7).

En este trabajo se ha utilizado una pantalla que no dispone de iluminación. En caso de disponer esta función se conectaría el pin 15 (LED +) en serie con una resistencia, para limitar la corriente, a una salida digital, y el pin 16 (LED -) a Tierra. La iluminación se activaría cuando la salida digital que conecta el pin 15 del LCD esté a nivel alto.

La pantalla dispone de señales para la habilitación/deshabilitación de funciones, como *R/W* que sirve para establecer el modo de escritura en el registro (para mostrar un

mensaje) o establecer el modo de lectura (para obtener el mensaje que se está mostrando), *Enable* que sirve para que el display muestre o no el mensaje aun estando alimentado, RS para seleccionar el registro de control o de datos. Para el control de estos dispositivos, el entorno Arduino incluye por defecto una librería llamada "*LiquidCrystall.h*" la cual implementa todas las instrucciones necesarias para controlar un LCD.

- **Experiencia**

Se han conectado los componentes como se describió anteriormente, y en el entorno Arduino se ha cargado el algoritmo Test_LCD.pde que se muestra a continuación:

```
#include <LiquidCrystal.h>

LiquidCrystal   lcd(4, 5, 6, 7, 8, 9);

void setup() {

  lcd.begin(16, 2);
  lcd.print("Test LCD");
}

void loop() {

  lcd.setCursor(0, 1);
  lcd.print(millis()/1000);
}
```

La primera línea de código incluye la librería *"LiquidCristal.h"* que ya tiene el entorno de Arduino por defecto. En la segunda línea se declara un dispositivo LCD; la estructura de esta sentencia es **NombreDisp(**RS,ENABLE,DB4,DB5,DB6,DB7**)** por lo que en nuestro caso hemos llamado al dispositivo *lcd*, y hemos configurado, entre otras, la salida digital 4 como RS y la 9 como DB7.

A continuación está el código correspondiente procedimiento de configuración. Aquí se inicia el dispositivo LCD mediante la línea **lcd.begin()**, indicando el número de caracteres por línea, y el número de líneas. En este caso, al ser el display 16 caracteres y

2 líneas, introducimos **lcd.begin**(16,2). La siguiente instrucción escribe en la primera posición de la primera línea "Test LCD".

El bucle principal consiste en posicionar el cursor en la primera posición de la segunda línea, y escribir el tiempo que ha transcurrido desde que se encendió el sistema. Al posicionar siempre el cursor en la primera posición, no es necesario limpiar la pantalla. Si por ejemplo se hiciese una cuenta desde un numero hasta cero, habría que limpiar la línea mediante la instrucción **lcd.clear()**,ya que al pasar por ejemplo de 100 a 99, se vería 990.

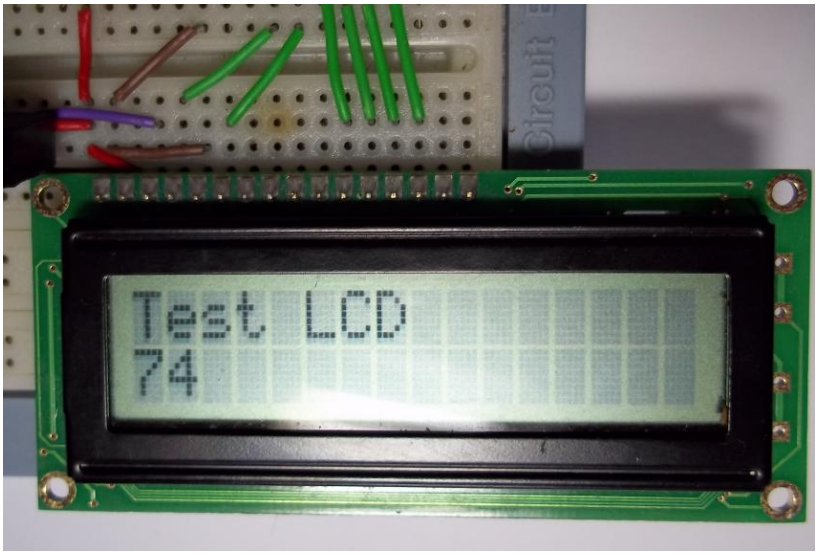


Imagen 5.1.5 – LCD funcionando con el algoritmo TEST LCD

Este algoritmo es útil para comprobar que se ha conectado el LCD correctamente con Arduino y que funciona

5.2. IMPLEMENTACIÓN DE BOTONES CON ARDUINO

Se han utilizado botones pulsadores de PCB en formato estándar con dos pines para poder controlar el sistema electrónico es necesario implementar pulsadores. Este tipo de botones pueden ser de dos tipos, con 4 pines o con 2 pines;

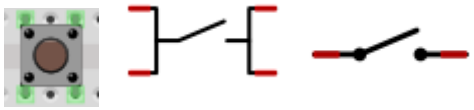


Imagen 5.2.1 – Botones en Fritzing

Por otro lado, destacar el uso que se le da en este apartado a las entradas analógicas de Arduino, convirtiéndolas en entradas digitales.

- **Esquema eléctrico del circuito**

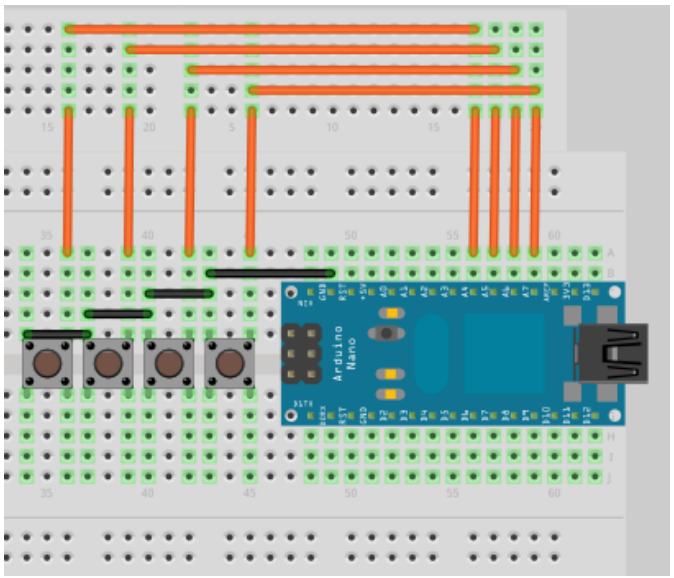


Imagen 5.2.2 – Vista del circuito implementado en Protoboard

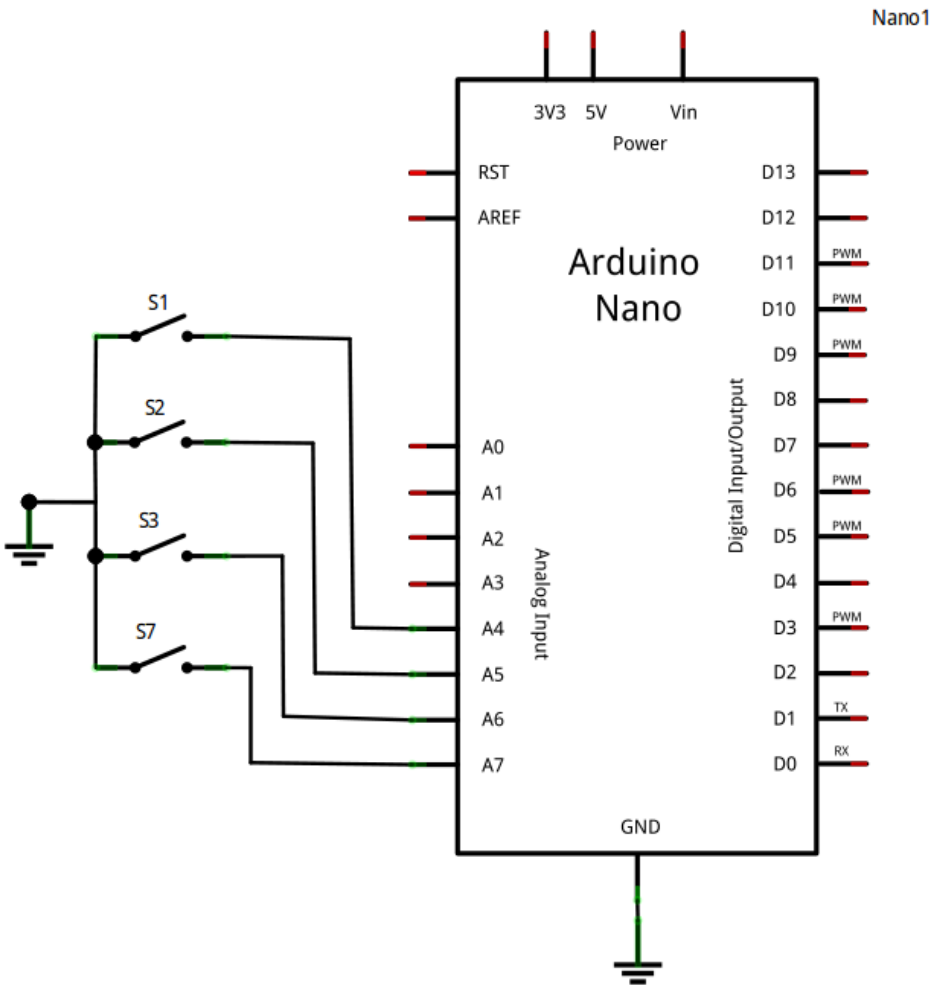


Imagen 5.2.3 – Esquemático del circuito implementado

- **Descripción del funcionamiento**

Las entradas analógicas pueden utilizarse como si fuesen entradas digitales, considerando que la entrada analógica A0 es la entrada digital 14. Para reducir el número de componentes necesarios en el circuito, se han utilizado las resistencias internas de pull-up del microcontrolador:

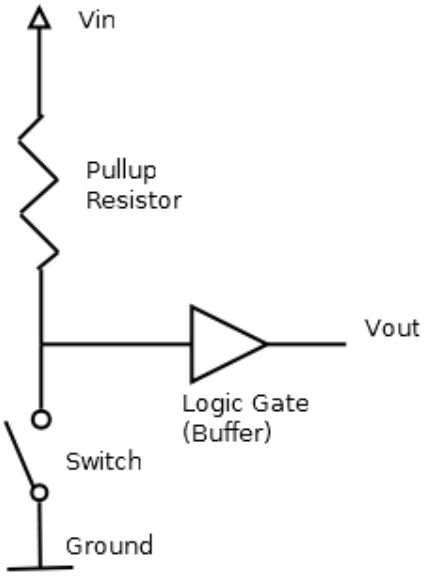


Imagen 5.2.4 – Esquema de pulsador con resistencia de pull-up

Suponiendo que el buffer impide el paso de corriente, cuando el interruptor está abierto, la resistencia de pull-up, presentará la misma tensión en ambos terminales (Vin). Sin embargo, cuando el botón se cierre, la entrada del buffer estará conectada directamente a tierra (Ground). En esta última situación aparecerá una corriente que circulara desde Vin hacia tierra, por lo que la resistencia de pull-up debe tener un valor elevado para que esta corriente sea pequeña. El valor de la resistencia interna de Arduino es de 20kΩ. Para saber cuándo se pulsa un botón, se lee la tensión en la salida del buffer (Vout). Cuando el botón se pulse estará a nivel bajo, y cuando esté abierto a nivel alto.

A cada botón que se implementa se le asignan 3 variables en el algoritmo:

- ✓ **BotonX**. Variable lógica en la que se almacena la lectura de la tensión en la entrada de la placa Arduino en la que se conecta el botón.
- ✓ **CBotonX**. Variable lógica que almacena la lectura de confirmación de la tensión en la entrada de la placa Arduino en la que se conecta el botón.
- ✓ **EABotonX**. Variable lógica que almacena el estado anterior del botón

Las dos primeras variables, BotonX y CBotonX, se usan para realizar la lectura del estado de la entrada. El motivo por el que se usan dos variables, es porque pueden aparecer lecturas erróneas debido a rebotes o en el botón o fluctuaciones en esa linea al pulsarlo o soltarlo. La siguiente imagen muestra esta situación:

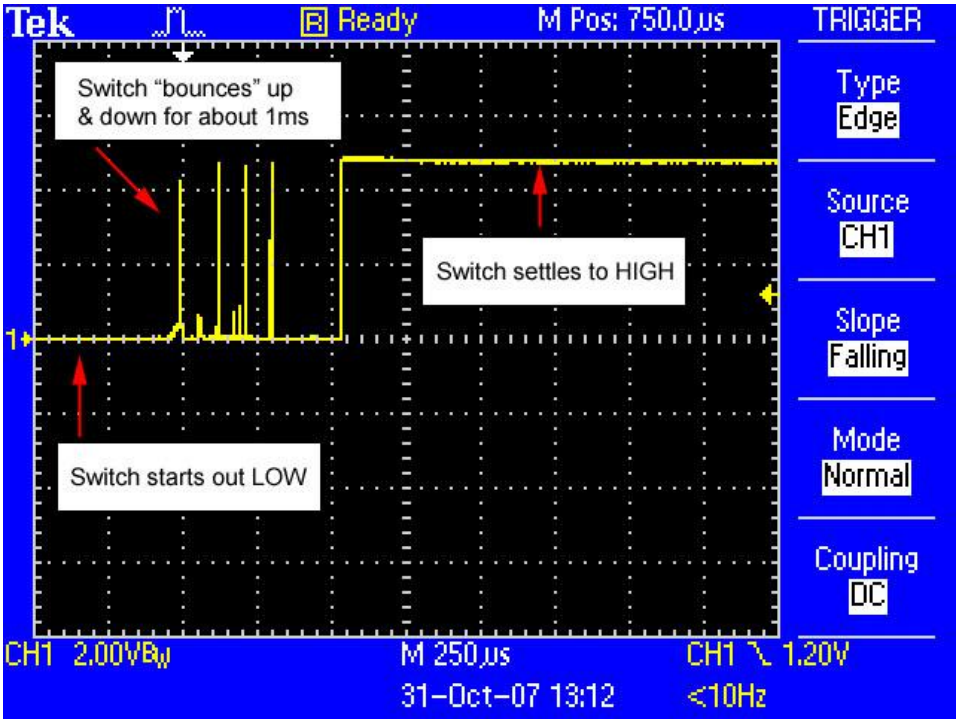


Imagen 5.2.5 – Rebote al pulsar un botón

La solución a esto es usar dos variables de manera que en una se almacene el estado al pulsar el botón y en otra se almacene el estado cuando pasa un intervalo de tiempo, suficiente para que ya no haya rebotes. Este retardo asegurará que si se pulsa el botón, el algoritmo vea que se pulsa el botón 1 vez, y no varias debido a las fluctuaciones. La tercera variable, EABotonX, se usa para almacenar el estado anterior y así confirmar que se ha producido una transición de estado en la línea del botón.

- **Experiencia**

Se han conectado los componentes como se describió anteriormente, y en el entorno Arduino se ha cargado el algoritmo Test_Botones.pde que se describe a continuación:

```
bool Boton1;   bool CBoton1;    bool EABoton1;
bool Boton2;   bool CBoton2;    bool EABoton2;
bool Boton3;   bool CBoton3;    bool EABoton3;
bool Boton4;   bool CBoton4;    bool EABoton4;


void setup()
{

  Serial.begin(19200);
  pinMode(14, INPUT);
  pinMode(15, INPUT);
  pinMode(16, INPUT);
  pinMode(17, INPUT);
  digitalWrite(14, HIGH);
  digitalWrite(15, HIGH);
  digitalWrite(16, HIGH);
  digitalWrite(17, HIGH);
}

void loop()
{
      Boton1 = digitalRead(17);  Boton2 = digitalRead(16);
      Boton3 = digitalRead(15);  Boton4 = digitalRead(14);

      delay(10);

      CBoton1 = digitalRead(17);  CBoton2 = digitalRead(16);
      CBoton3 = digitalRead(15);  CBoton4 = digitalRead(14);

      if ((Boton1 == CBoton1)&&(Boton1 != EABoton1)&&(Boton1 == LOW))
      {
            Serial.println("Se ha pulsado el boton 1");
      }
      if ((Boton2 == CBoton2)&&(Boton2 != EABoton2)&&(Boton2 == LOW))
      {
            Serial.println("Se ha pulsado el boton 2");
      }
      if ((Boton3 == CBoton3)&&(Boton3 != EABoton3)&&(Boton3 == LOW))
      {
            Serial.println("Se ha pulsado el boton 3");

      }
      if ((Boton4 == CBoton4)&&(Boton4 != EABoton4)&&(Boton4 == LOW))
      {
            Serial.println("Se ha pulsado el boton 4");
      }

      EABoton1 = Boton1;  EABoton2 = Boton2;
      EABoton3 = Boton3;  EABoton4 = Boton4;
}
```

El algoritmo se inicia con la declaración de variables. En este caso se han usado 4 botones, por lo que se declaran 12 variables. A continuación aparece el procedimiento de configuración. Aquí se establece la comunicación por el puerto serie para poder ver

los resultados en la pantalla del ordenador. Se configuran las salidas digitales en el rango 14-17 y se activa la resistencia de pull-up mediante la instrucción **digitalWrite(**NumeroEntrada,HIGH**)**.

El funcionamiento del bucle principal consiste en leer el estado de los botones y almacenar la lectura de cada uno en la variable correspondiente. A continuación se introduce una espera de 10 milisegundos y se vuelve a realizar el mismo procedimiento pero almacenando la lectura en otra variable. Una vez que ya se tienen los datos de los eventos ocurridos, se evalúan tres posibilidades para cada botón:

- ✓ Si coinciden las dos variables con la lectura de cada botón (BotonX y CBotonX, para evitar falsas lecturas)
- ✓ Si ha habido un cambio de estado en la lectura del botón (Cuando BotonX es diferente de EABotonX) y
- ✓ Si el estado es "LOW" (que significa que se está pulsando).

Si las tres se cumplen, entonces se realizarán las acciones programadas; mostrar por pantalla un mensaje que dice "Se ha pulsado el botón numero X". Finalmente se guarda el estado de cada botón en la variable correspondiente (EABotonX) empieza de nuevo el bucle. De esta forma se ha implementado satisfactoriamente 4 pulsadores en Arduino

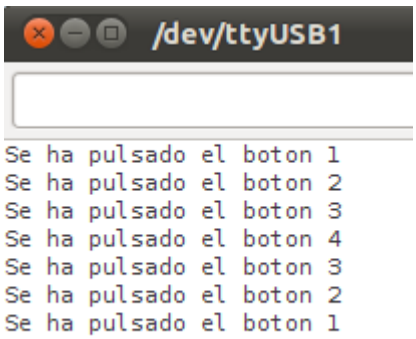


Imagen 5.2.6 - Visualización de los resultados por el puerto serie

5.3. IMPLEMENTACIÓN DEL MENÚ DE SELECCIÓN

Dado que el dispositivo de medida que se está diseñando en este trabajo tiene varios modos de operación, ha sido necesario implementar una interfaz de usuario, la cual consiste en la visualización de mensajes que informen en qué situación se encuentra el aparato y muestre los resultados de las medidas. En el apartado anterior se explicó el procedimiento de implementación de cuatro botones pulsadores. Usando ese mismo circuito y modificando el algoritmo anterior es posible implementar un menú conductor que servirá para que el usuario seleccione la operación que desea realizar.

- **Descripción del funcionamiento**

La interfaz gráfica la constituyen tres bloques:

Imagen 5.3.1 – Diagrama de funcionamiento

La entrada de datos puede realizarse desde el ordenador (símbolo UISB) o mediante pulsadores. Arduino permite la implementación de comunicaciones con el ordenador usando el teclado mediante el puerto serie. De esta forma se puede usar la pantalla del ordenador para representar la información, lo que supone una ventaja añadida a la hora de mostrar los datos, ya que mediante otro software compatible con Arduino, como por ejemplo Processing, podemos implementar gráficas que muestren estos datos, en lugar de usar texto alfanumérico. Esta opción es muy interesante, pero restringe el uso del dispositivo a lugares en los que haya un ordenador; por este motivo se ha optado por usar pulsadores para obtener los datos de entrada. Para mostrar los datos se presenta el mismo problema; el dispositivo no puede operar sin un ordenador.

El estado del dispositivo se controla mediante Arduino. El procedimiento de control consiste en una variable que almacena un valor, que corresponderá al estado

codificado. A cada valor se le asigna un mensaje para mostrar en pantalla, de manera que en todo momento el usuario sabrá en qué estado se encuentra el dispositivo. Este valor se modifica mediante los pulsadores, y cada uno lo modifica de una forma; los botones Izquierda y Derecha restan y suman 1 respectivamente, el botón Ok multiplica por 10 y el botón Atrás divide entre diez. A continuación se muestra un ejemplo de la evolución del valor de la variable que modificamos (entre paréntesis) en función del menú que queremos visualizar:

1. Operación (1)
  1.1. Operación (10)
    1.1.1. Operación (100)
    1.1.2. Operación (101)
  1.2. Operación (11)
  1.3. Operación (12)
    1.3.1. Operación (120)
    1.3.2. Operación (121)
  1.4. Operación (13)
2. Operación (2)
3. Operación (3)
  3.1. Operación (30)
  3.2. Operación (31)

Podemos movernos por un menú inicial de 3 opciones, y el valor mínimo de la variable de control (a partir de ahora se le llamará "Opción") es 1. Mediante los botones Izquierda/Derecha podemos movernos por las opciones 1, 2, 3. Cuando se pulsa el botón Ok, se multiplica la variable "Opción" por 10. Esto significa que si por ejemplo, estamos en la opción 1, pasaremos al submenú 1.1, y con los botones Izquierda/Derecha podemos movernos igual que anteriormente. En caso de querer regresar al menú anterior, con el botón Atrás dividimos el valor de "Opción" entre diez, y nos quedamos con la parte entera, de esta forma volveremos al menú anterior.

La siguiente tabla recoge la correspondencia entre el mensaje de la operación que se debe mostrar y el valor que contiene la variable "Opción". También se indican que botones pueden modificar el valor en cada situación:

Mensaje de Operación	"Opción"	Botones permitidos
Operación 1	1	Der/Ok
Operación 2	2	Izq/Der/Ok
Operación 3	3	Izq/Ok
Operación 1.1	10	Der/Ok/Atrás
Operación 1.2	11	Izq/Der/Ok/Atrás
Operación 1.3	12	Izq/Der/Ok/Atrás
Operación 1.4	13	Izq/Ok/Atrás
Operación 1.1.1	100	Der/Ok/Atrás
Operación 1.1.2	101	Izq/Ok/Atrás
Operación 1.3.1	120	Der/Ok/Atrás
Operación 1.3.2	121	Izq/Ok/Atrás
Operación 3.1	30	Der/Ok/Atrás
Operación 3.2	31	Izq/Ok/Atrás

Por ejemplo, si estamos con el menú en la Operación 1, solo se modificará la variable "Opción" con los botones Derecha (+1, y pasaríamos a Operación 2) o Ok (multiplica por 10 y pasa a la Operación 1.1). Si por ejemplo estamos en la opción 2 y pulsamos Ok, "Opción" pasará a contener el valor "20", lo que significa que hay que realizar la tarea a la cual se le asigna un código. Por ello, el algoritmo necesita poder decidir cuándo parar de leer los botones para realizar la tarea ordenada. La siguiente tabla muestra el valor de la variable "Opción" que significa que ha de realizarse una tarea. A cada operación le corresponde un valor:

Nombre de Operación	Valor de "Opción"
Operación 1.1.1	1000
Operación 1.1.2	1010
Operación 1.2	110
Operación 1.3.1	1200
Operación 1.3.2	1210
Operación 1.4	130

Operación 2	20
Operación 3.1	300
Operación 3.2	310

Es importante acotar el funcionamiento de los botones, de esta forma se evitará que el sistema entre en un estado no programado. En la siguiente tabla se muestra, en el caso de los botones Izquierda y Derecha, que código debe tener la variable "Opción" para que estos la modifiquen, y en el caso de Ok y Atrás cuando no deben modificarla:

Izquierda	2,3,11,12,13, 101, 121,31
Derecha	1,2,10,11,12, 30,100,120
Ok (Cuando no modifica)	20,110,130,300,310 1000,1010,1200,1210
Atrás (Cuando no modifica)	1,2,3

- **Experiencia**

Se ha implementado el circuito con pulsadores mostrado en el apartado 5.2, y se ha modificado el código para realizar un menú como el descrito en el punto anterior. El algoritmo tiene el nombre de TEST_MENU.pde. Se ha modificado el nombre del botón por la función que realiza, y se ha declarado una variable global llamada "Selección", que contiene el estado codificado.

```
bool Izquierda;   bool CIzquierda;    bool EAIzquierda;
bool Derecha;      bool CDerecha;      bool EADerecha;
bool Ok;           bool COk;           bool EAOk;
bool Salir;        bool CSalir;        bool EASalir;

int Seleccion;
```

Para realizar la muestra del mensaje correspondiente a cada estado, se ha programado un subprograma al que se le introduce un código, y escribe en pantalla el mensaje que corresponde. En este caso se ha usado el puerto serie para usar menos componentes en la experiencia.

```
void EscribirMensaje(int Opcion)
{
  switch(Opcion)
  {
    case 1:
    Serial.println("Esta usted en la opcion 1");
    Serial.println("       >      OK    ATRAS");
    break;

    case 2:
    Serial.println("Esta usted en la opcion 2");
    Serial.println("<      >      OK    ATRAS");
    break;

    case 3:
    Serial.println("Esta usted en la opcion 3");
    Serial.println("<             OK    ATRAS");
    break;

    case 10:
    Serial.println("Esta usted en la opcion 1.1");
    Serial.println("       >      OK    ATRAS");
    break;
```

Esta parte del código corresponde a la cabecera del subprograma y a parte de la estructura de selección. El procedimiento **EscribirMensaje()** recibe como parámetro un valor entero, el cual se evalúa mediante una estructura de selección, y en función del valor que contenga se muestra un mensaje diferente. El mensaje consiste en dos líneas; la primera informa del estado del dispositivo y la segunda de los botones disponibles para pulsar. Si por ejemplo estamos en la opción 1 y pulsamos el botón de Derecha o el botón Atrás no sucederá nada.

```
case 120:
Serial.println("Esta usted en la opcion 1.3.1");
Serial.println("       >      OK    ATRAS");
break;

case 121:
Serial.println("Esta usted en la opcion 1.3.2");
Serial.println("<             OK    ATRAS");
break;

case 20:
Serial.println("Realizando la operacion 2");
Serial.println("                    ATRAS");
break;

case 130:
Serial.println("Realizando la operacion 1.2");
Serial.println("                    ATRAS");
break;
```

El procedimiento de configuración es igual que el programado en el apartado 5.2, con la diferencia de que en este se asigna un valor a la variable "Selección" y se hace una llamada al procedimiento **EscribirMensaje()** para mostrar la primera opción.

```
void setup()
{

  Serial.begin(19200);
  pinMode(14, INPUT);
  pinMode(15, INPUT);
  pinMode(16, INPUT);
  pinMode(17, INPUT);
  digitalWrite(14, HIGH);
  digitalWrite(15, HIGH);
  digitalWrite(16, HIGH);
  digitalWrite(17, HIGH);
  Seleccion = 1;
  EscribirMensaje(Seleccion);
}
```

El bucle principal también es igual que el del apartado 5.2. La modificación consiste en una línea de selección que sirve para asegurar que no se pasan a estados no programados; por ejemplo si el valor de Selección es menor que 10, no se dividirá el convenido de esta variable aunque se pulse el botón Atrás:

```
if ((Izquierda == CIzquierda)&&(Izquierda != EAIzquierda)&&(Izquierda == LOW))
{
    if ((Seleccion==2)||(Seleccion==3)||((Seleccion>=11)&&(Seleccion<=13))
        ||(Seleccion==101)||(Seleccion==121)||(Seleccion==31))
    {
      Seleccion--;
      EscribirMensaje(Seleccion);
    }
}
if ((Derecha == CDerecha)&&(Derecha != EADerecha)&&(Derecha == LOW))
{
    if ((Seleccion==1)||(Seleccion==2)||((Seleccion>=10)&&(Seleccion<=12))
         ||(Seleccion==100)||(Seleccion==120)||(Seleccion==30))
    {
      Seleccion++;
      EscribirMensaje(Seleccion);
    }
}
```

```
if ((Ok == COk)&&(Ok != EAOk)&&(Ok == LOW))
{
    if ((Seleccion<=121)&&(Seleccion!=20)&&(Seleccion!=110)&&(Seleccion!=130))
    {
      Seleccion *= 10;
      EscribirMensaje(Seleccion);
  }
}
if ((Salir == CSalir)&&(Salir != EASalir)&&(Salir == LOW))
{
    if (Seleccion>=10)
  {
      Seleccion /= 10;
      EscribirMensaje(Seleccion);
  }
}
```

Finalmente se ha cargado el algoritmo en Arduino, y se ha comprobado el funcionamiento del menú. El algoritmo funciona correctamente y "no hay falsas pulsaciones":

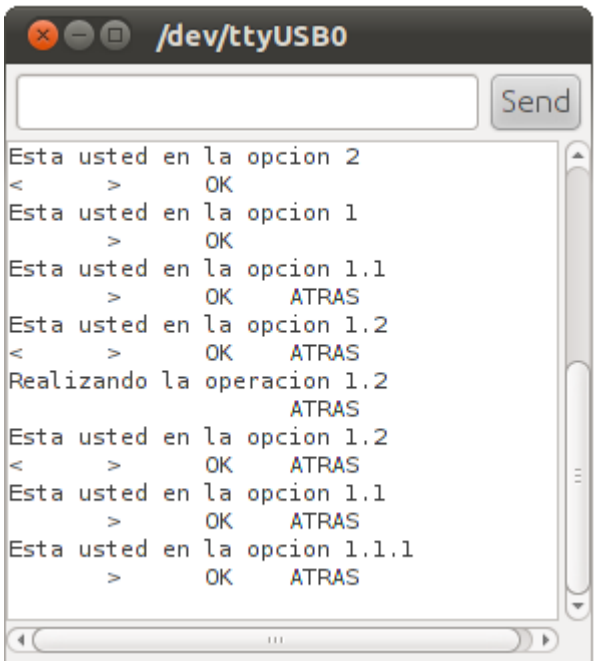


Imagen 5.3.2 – Menú conductor implementado

5.4. IMPLEMENTACIÓN DEL BUS SPI

Arduino y ADE7759 se comunican mediante un bus de comunicaciones SPI. En el anexo 1; Documentación de ADE7759, y en el anexo 3; Comunicaciones SPI, hay información sobre el funcionamiento de la comunicación SPI para cada dispositivo en particular.

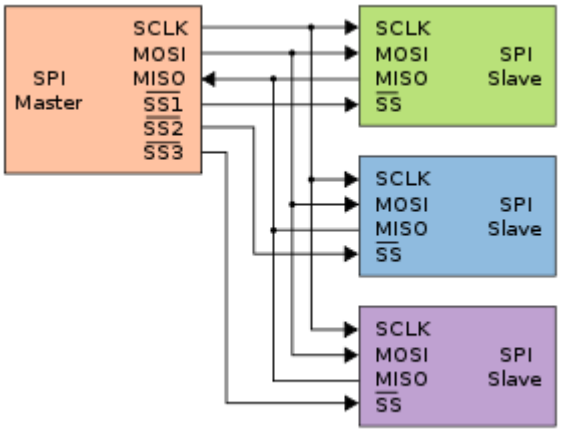


Imagen 5.4.1 - Representación de bus SPI con un maestro y tres esclavos

El bus de comunicación SPI está formado por 4 líneas DIN, DOUT, SCLK Y CS. Mediante la línea DIN se envía información desde el maestro al esclavo, y con DOUT en sentido inverso. La información ser transmite en paquetes de 1 byte, de manera que si queremos enviar un paquete de 3 bytes el proceso consistirá en enviar 3 veces 1 byte. La línea SCLK es la señal de sincronización para el envío de datos; aquí habrá una señal cuadrada que genera el maestro. La línea CS se activa a nivel bajo, y el bus contiene tantas como dispositivos esclavo haya conectados al bus. Esta señal es la que habilita al dispositivo esclavo para recibir o enviar datos, o lo deshabilita para enviarlos o recibirlos de otros dispositivos del bus.

En un bus de comunicación están conectados dos tipos de dispositivos, el maestro y los esclavos. El maestro es el que controla el bus de datos, mientras que los esclavos solo aportan información cuando la solicita el maestro. También es el que controla las señales CS. El protocolo de comunicación SPI es muy variado; con el mismo bus de

datos podemos enviar información de diferentes formas como por ejemplo enviar primero el LSB o el MSB cuando el dato es mayor que un byte, o sincronizar la transmisión de datos en el flanco de bajada o de subida SCLK. Entre estas características están:

- Si los bits más significativos van delante o detrás (MSB o LSB).
- Si los datos se leen en el flanco de bajada o de subida del reloj.
- La velocidad máxima a la que puede trabajar el dispositivo.
- Si habilitamos interrupciones.
- El nivel lógico del reloj cuando está inactivo.

Arduino ya viene preparado para las comunicaciones SPI. Las líneas SCLK, DIN y DOUT están determinadas por el microcontrolador, la de selección del dispositivo se puede asignar a cualquiera de las restantes. El microcontrolador que utiliza Arduino tiene una serie de registros, que utiliza para la comunicación SPI. Estos registros tienen un nombre reservado por el sistema, por lo que no necesitamos definirlos en nuestro programa:

Nombre	**Función**	**Descripción**
SPCR	SPI Control Register	Configura las comunicaciones
SPDR	SPI Data Register	Almacena los datos a enviar y recibidos
SPSR	SPI Status Register	Indica estado de las comunicaciones

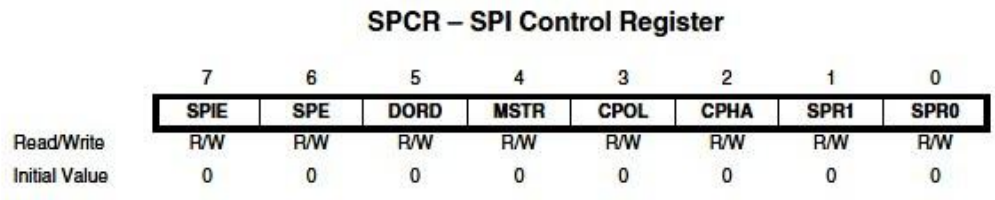


Imagen 5.4.2 – Composición de los bits en el registro de control de comunicaciones SPI

A continuación se muestra un diagrama del registro de control de la comunicación SPI, en el que se puede ver la función los bits que componen este registro:

SPIE	Habilita interrupciones cuando es 1
SPE	Habilita SPI cuando es 1
DORD	Determina el orden de envío de los bits: • Con 1primero el LSB • Con 0 primero el MSB
MSTR	Con "1" Arduino es Maestro y con 0 Esclavo
CPOL	Con 1 el reloj esta a nivel alto cuando está inactivo, y con 0 a nivel bajo.
CPHA	Con 1 los datos se toman en el flanco de bajada y con 0 en el de subida.
SPR1 y SPR0	Establecen la velocidad de transmisión : • 00 es la más rápida (4MHz) • 11 es la más lenta (250KHz)

El entorno de programación de Arduino incluye una librería que proporciona funciones para la transferencia de datos mediante SPI. El interface SPI se inicia automáticamente cuando se incluye esta librería en el algoritmo.

- **Esquema eléctrico**

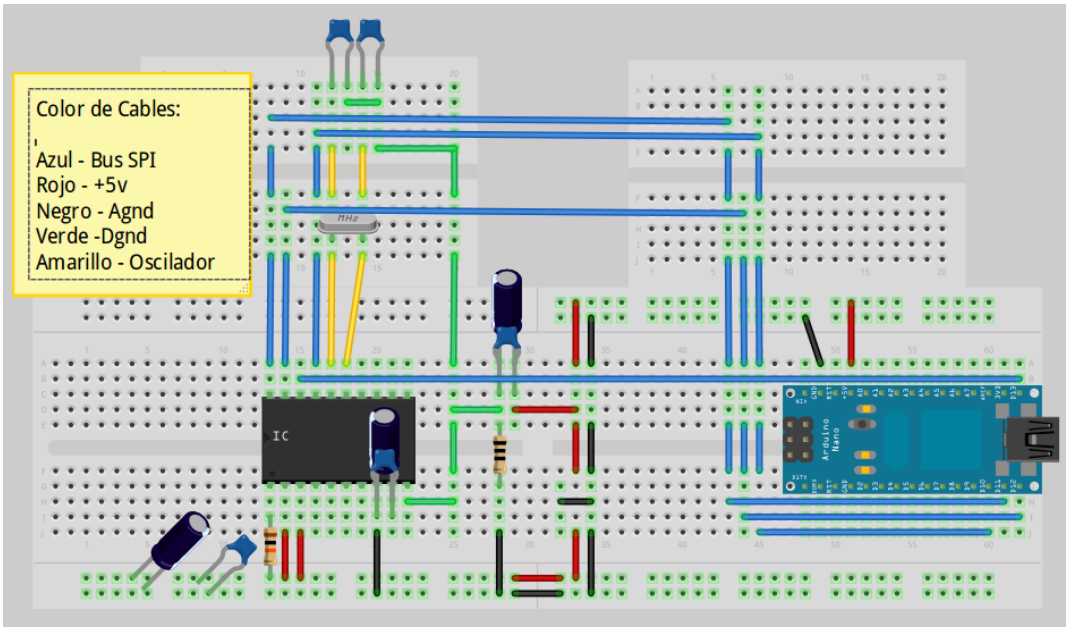


Imagen 5.4.3 - Vista en la placa de prototipado de ADE7759 con Arduino Nano

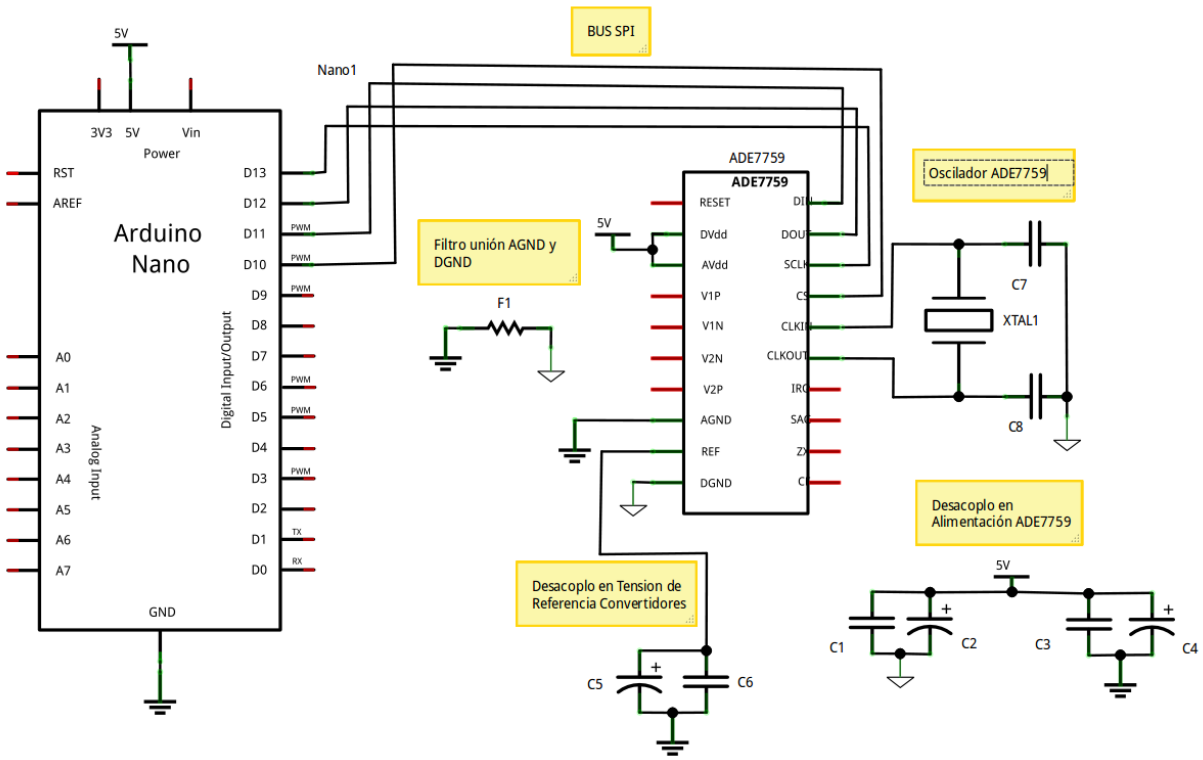


Imagen 5.4.4 - Esquema de conexiones ADE7759 y Arduino

- **Descripción del funcionamiento**

Para alimentar ADE7759 se usa la salida regulada de 5V que tiene Arduino. Se han conectado los condensadores de desacoplo con los valores recomendados por el fabricante en el datasheet. Se puede observar que ADE7759 tiene dos entradas de alimentación AVDD y DVDD; esto se debe a que el circuito integrado está compuesto por dos partes, la analógica y la digital. Esto implica que en el circuito tenemos dos tipos de tierra; la tierra analógica y la tierra digital. La tierra analógica se conecta directamente entre GND en Arduino y AGND en ADE7759.

La tierra digital, que en ADE7759 es DGND, está conectada a GND en un punto mediante un filtro de ferrita como recomienda Analog Devices. El la función de este filtro es la de evitar que se introduzca ruido en el circuito durante el proceso de muestreo, como consecuencia del proceso de cuantización en los convertidores- En el esquema anterior aparece con el símbolo de una inductancia conectando AGND y DGND en un punto del circuito.

Este bus SPI está compuesto por dos elementos, el maestro que es la placa Arduino Nano, y el esclavo que es ADE7759. Para configurar la comunicación es necesario conocer los datos característicos del integrado:

- ✓ Cualquier operación, de lectura o escritura comienza enviando un código en el que se indica el tipo de operación y el registro en el que se opera. Si la operación de escritura enviaremos el bit más significativo primero, y los 7 bits restantes la dirección del registro a escribir. Si la operación es de lectura el primer MSB será un 0 seguido de 7 bis indicando la dirección del registro a leer.
- ✓ La velocidad máxima de comunicación es de 3,33 MHz.
- ✓ Cuando se transmite información (en ambos sentidos) primero se envía el bit más significativo. En caso de ser una palabra multibyte, se empieza enviando el byte más significativo.
- ✓ Cuando se transmiten datos estos se colocan en el bus en sincronía con el flanco de subida de SCLK, y se transmiten en el flanco de bajada.

- ✓ Antes de empezar la operación hay que habilitar al dispositivo, poniendo CS a nivel bajo. El tiempo de espera es de 20nS antes de poder empezar a operar.
- ✓ El tiempo de espera entre transferencia de bytes es 4µS

- **Experiencia**

A continuación se describe el código programado en el entorno de Arduino para implementar la comunicación con ADE7759. Para hacer más simple la lectura del código, este se ha separado en diferentes pestañas. La primera pestaña contiene la declaración de variables, el procedimiento de configuración y el bucle principal. La segunda pestaña contiene los subprogramas y funciones. El código que contiene el algoritmo principal "Test_SSPI.pde" corresponde a la declaración del bus SPI y variables globales, el procedimiento de configuración y el bucle principal.

```
#include <SPI.h>

#define DIN 11
#define DOUT  12
#define SPICLOCK  13
#define CHIPSELECT 10

byte LINECYC = 0x13;
byte WRITE = 0b10000000;
```

Comienza incluyendo la librería "*SPI.h*" y determinando la salida digital de corresponde a cada línea del bus SPI. El microcontrolador usa por defecto las salidas 11, 12 y 13 para las líneas DIN, DOUT y SCLK, y no pueden ser otras. En cambio, CS sí puede ser elegida, y conectarla donde resulte más beneficioso en el diseño. También se ha declarado una variable global de un byte; LINECYC que contiene la dirección del registro donde se va a realizar el test.

```
void setup()
{
  Serial.begin(19200);
  pinMode(DIN, OUTPUT);
  pinMode(DOUT, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(CHIPSELECT,OUTPUT);
  digitalWrite(CHIPSELECT,HIGH);
  SPCR = (1<<SPE)|(1<<MSTR)|(1<< CPHA);
  Serial.println ("Comenzamos ");
  delay(10);
}
```

El procedimiento de configuración inicia el puerto serie para poder obtener por pantalla los resultados de esta prueba. También se configura el registro SPCR (Registro de Control SPI. Cada bit de este registro tiene un nombre determinados por el fabricante del microcontrolador, y tanto este como el del registro están reservados, por lo que no hay que declararlos, ni se puede usar para otra función con ese nombre El contenido de este registro se asigna con la línea SPCR = ((1<<SPE)|(1<<MSTR)|(1<<CPHA). Con ella establecemos a nivel alto los bits los bits SPE (habilitar la comunicación SPI), MSTR (establece Arduino como maestro en el bus), CPHA (Determina que los datos se adquieren en el flanco de bajada) y SPR0 (para usar la segunda velocidad más alta, ya que la primera es demasiado alta). Las señales del bus se configuran mediante la instrucción **pinMode()**, y para CS (o cualquier línea que lo necesite) activamos la resistencia de pull-up mediante **digitalWrite()**.

```
void loop()
{
  int Lectura = 0;

  for(int i=0; i<=300; i++)
 {
    Lectura = LeerRegistro(LINECYC,2);
    Serial.print ("Dato almacenado previamente: ");
    Serial.println (Lectura, HEX);

    Serial.print ("Se va a escribir el valor: ");
    Serial.println (i, HEX);
    EscribirRegistro(LINECYC,i);
    Serial.println ("Escritura finalizada");
 }
}
```

El bucle principal contiene un sencillo código para comprobar que la comunicación funciona. El registro que estamos usando para esta prueba contiene X bits, y es de Lectura/Escritura, por lo tanto, se puede escribir un valor, y después leerlo para comprobar que la operación se realiza correctamente. Para facilitar el proceso de detección de errores en caso que los haya, se está utilizando el puerto serie para ver en pantalla el dato que se lee y el dato que se escribe. La operación programada en esta prueba consiste en leer el contenido del registro, mostrarlo en pantalla y escribir un nuevo valor.

Se ha declarado una variable local (Lectura) que sirve para volcar en ella el contenido que leemos del registro y se ha declarado una variable (i), que sirve de contador en la estructura "for" y a la vez es contiene el valor que queremos guardar. Mediante un bucle "for" se repite la operación 300 veces. La operación se realiza en dos pasos (dentro del bucle "for"); primero se lee el valor que contiene el registro de ADE7759 y en segundo lugar escribimos un nuevo valor (valor del contador). Esta operación se apoya en la función **LeerRegistro()** y el subprograma **EscribirRegistro()**, que están programados en la pestaña Funciones_SPI.pde

```
long long LeerRegistro(byte EsteRegistro, int BytesParaLeer)
{
  long long Resultado = 0LL;
  byte DatoParaEnviar = 0;

  digitalWrite(CHIPSELECT, LOW);

    DatoParaEnviar = EsteRegistro;
    SPI.transfer(DatoParaEnviar);
    delayMicroseconds(4);

    while(--BytesParaLeer >= 0)
    {
      Resultado |= SPI.transfer(0x00);

      if (BytesParaLeer > 0)
      {
        Resultado = Resultado << 8;
      }

      delayMicroseconds(4);
    }

   digitalWrite(CHIPSELECT, HIGH);
   return(Resultado);
}
```

LeerRegistro() es una función que devuelve un numero con el contenido de la lectura. Cuando se llama a esta función se introducen dos parámetros, la dirección del registro que queremos leer y el número de bytes que lo forman

En esta función se usan 2 variables locales. Dos son de tipo byte; "datoRecibido" donde se almacena el byte que recibimos desde ADE7759 y "datoParaEnviar" que contiene el byte que vamos a enviar por el bus SPI. Para preparar ADE7759 para la comunicación, el primer paso es habilitar la línea CS poniéndola a nivel bajo y esperar 1 microsegundo para que el sistema se estabilice. Es muy importante respetar las condiciones de temporización de ADE7759, por lo que siempre que se modifique la línea CS hay que esperar 1 microsegundo, y esperar 4 microsegundos cuando se envíen datos por el bus. A continuación se envía la dirección del registro (y espera de 4 microsegundos para que la información esté disponible.

A la variable Resultado se le ha asignado el primer byte que recibamos por el puerto SPI, que se escribirá en el LSB de la variable "Resultado" pero será el MSB de la lectura si esta es de más de un byte. En caso de leer 2 bytes, el contenido de "Resultado" se desplazará a la izquierda 8 bits antes de leer un byte, liberando los LSB de la variable "Resultado" para almacenar ahí el siguiente. Finalmente se configura la señal CS a nivel bajo y se devuelve la variable "Resultado" con el contenido de la lectura.

```
void EscribirRegistro(byte esteRegistro, int esteValor)
{
  esteRegistro = WRITE | esteRegistro;
  byte datoParaEnviar = esteRegistro;

  digitalWrite(CHIPSELECT, LOW);
  delayMicroseconds(1);

  SPI.transfer(datoParaEnviar);
  delayMicroseconds(4);

  datoParaEnviar = highByte(esteValor);
  Serial.print("Enviando MSB:   ");
  Serial.println(datoParaEnviar,HEX);
  SPI.transfer(datoParaEnviar);
  delayMicroseconds(4);

  datoParaEnviar = lowByte(esteValor);
  Serial.print("Enviando LSB:   ");
  Serial.println(datoParaEnviar,HEX);
  SPI.transfer(datoParaEnviar);
  delayMicroseconds(4);

  digitalWrite(CHIPSELECT, HIGH);
}
```

A diferencia de la lectura, la escritura no es una función, sino un procedimiento. **EscribirRegistro()** recibe dos parámetros; la dirección del registro en el que tiene que escribir y el valor para escribir. En la primera línea del código se puede observar que se hace un enmascaramiento con la dirección del registro. "WRITE" contiene el valor para enmascarar la dirección, pero en caso de querer optimizar el código resulta ventajoso eliminarla como variable local y escribir su valor directamente en el enmascaramiento. Las direcciones de los registros están contenidas en bytes, pudiendo tener un valor comprendido desde 0 hasta 256 (en decimal). Sin embargo, ADE7759 tiene 23 registros empezando desde el 0 hasta el 23. Esto significa, que el bit más significativo de la dirección siempre va a ser 0. Este bit es el que se usa para determinar la operación, por eso cuando realizamos una lectura lo dejamos a 0, y si se realiza una escritura, se suma la dirección del registro con el numero binario 10000000, de modo que el valor resultante es un byte que enviamos a ADE7759 y este interpreta que vamos a escribir el registro. También se puede ver que se ha declarado una variable local que contendrá el byte que hay que enviar por el bus SPI en cada instante.

El procedimiento de escritura comienza poniendo a nivel bajo la señal CS. En primer lugar se envía la instrucción de escritura junto con la dirección del registro. Se espera 4 microsegundos para que el registro de ADE7759 esté preparado y se carga en la variable "datoParaEnviar" el MSB del valor que queremos escribir. Se envía igual que se envió el byte de instrucción y se esperan otros 4 milisegundos. A continuación se carga en la variable "datoParaEnviar" el LSB byte del valor que queremos escribir y se espera 4 microsegundos. Para finalizar la escritura, se deshabilita la señal CS.

El algoritmo se ha cargado en Arduino, conectando los componentes como se expuso anteriormente. Las siguientes imágenes muestran los resultados obtenidos:.

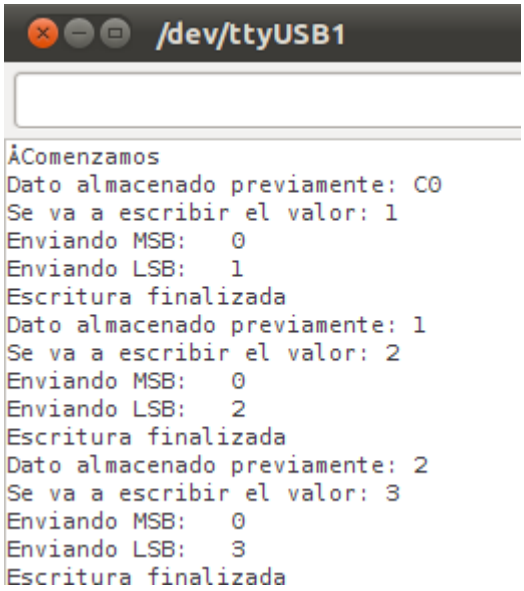


Imagen 5.4.5 – Resultado de Test_SPI con 1 byte

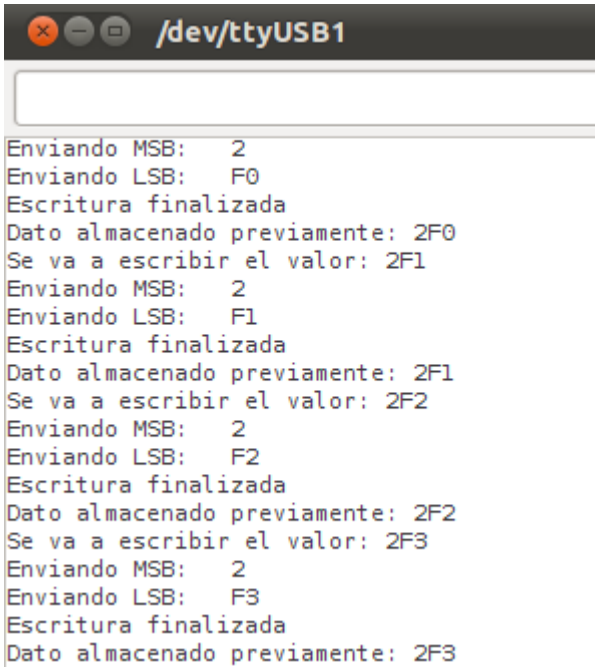


Imagen 5.4.6 – Resultado de Test_SPI con 2 byte

En la primera imagen se puede ver como se escriben valores de 1 byte, por lo que el MSB siempre es 0. Sin embargo en la segunda imagen se están enviando 2 byte, y el primer byte es 2. También se confirma que el resultado leído es el último valor escrito.

5.5. IMPLEMENTACION DEL ALGORITMO DE CALCULO RMS

Este apartado se explica cómo se ha implementado en el entorno de programación Arduino el algoritmo para el cálculo de valores eficaces que se expone en el apartado 4.1. Cálculo de valores eficaces. Este circuito implementado y el algoritmo programado se apoyan en los usados en el apartado 5.4. Implementación del bus SPI para realizar la comunicación con ADE7759.

La siguiente imagen muestra la implementación de la red de entrada al canal 2 de ADE7759. R1 y R2 forman junto con C1 y C2 respectivamente los filtro antialiasing. El valor de los condensadores es de 33nF, y de las resistencias 1kΩ. La resistencia R3 es de 510kΩ y sirve para hacer un divisor de tensión junto con R2:

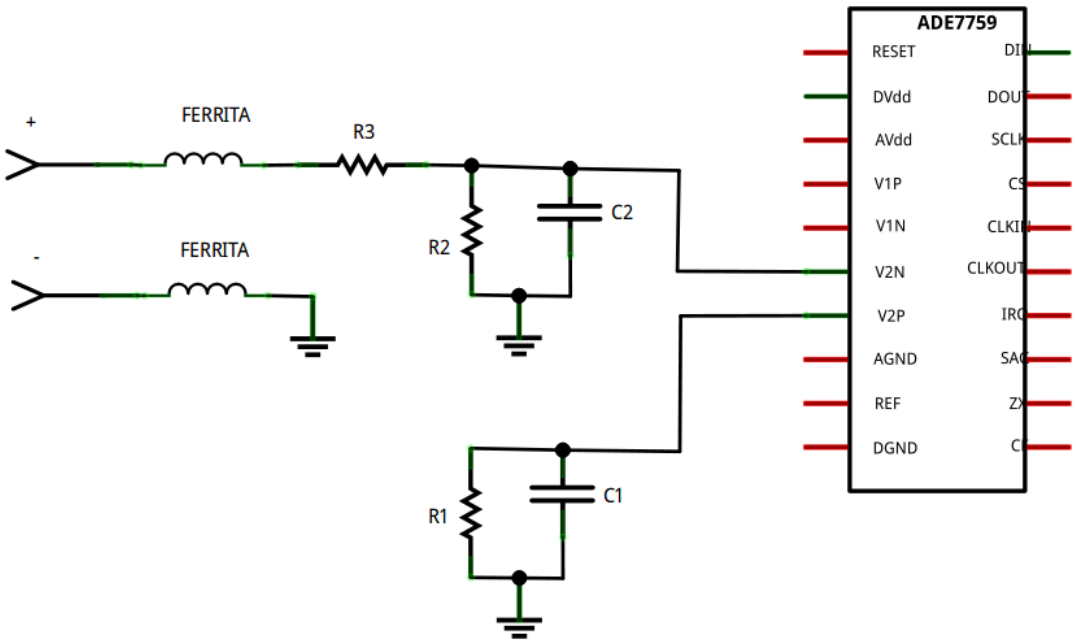


Imagen 5.5.1 – Implementación de la red de entrada al canal 2

Para facilitar la legibilidad del código, se aplica programación modular, programando cada operación en un subprograma. El algoritmo se incluye en el trabajo con el nombre "TEST_RMS.pde"

- **Post-Procesamiento**

A continuación se muestra el código correspondiente a la función que adapta la muestra que recibimos desde ADE7759 a una variable de Arduino:

```
long AdaptarMuestra(long MuestraEntrante)
{
  bool signo = LOW;
  long Resultado = 0L;

  signo = bitRead(MuestraEntrante,19);

  Resultado = MuestraEntrante >> 7;

  if (signo == LOW)
  {
    Resultado &= 0x00000FFF;
  }else{
    Resultado |= 0xFFFFF000;
  }
  return(Resultado);
}
```

Esta función tiene dos parámetros de entrada: la palabra leída desde el registro WAVEFORM de ADE7759 contenida en una variable "long", y un número entero que indica el número de LSB que se desechan. La función tiene declarada variable local de 4 bytes en la que se almacenará el resultado que debe devolver la función. La primera operación que realiza la función es la de desplazar el contenido de la palabra de entrada el numero de bits deseado y almacenarlo en la variable Resultado, eliminando de este modo los LSB de la muestra. A continuación se lee el signo, y en función de si es positivo o negativo se opera la variable Resultado para que el valor que contenga sea el valor real que indicaba la muestra que se recibió. En caso de ser positiva, se dejan los LSB como quedan, y se asegura que estarán a 0 todos los bits restantes, a partir del bit que indicaba el signo de la muestra. Esto se hace mediante una operación "AND" a nivel de bits. En caso de que sea positiva, la operación consistiría en rellenar con '1' estos bits. Para ello se usa la operación "OR" a nivel de bits.

Para implementar el algoritmo de cálculo de valores eficaces, es necesario implementar un filtro IIR. Observando el diagrama de flujo del filtro resulta fácil diseñarlo. Esta función es un filtro de respuesta infinita a impulso, que devuelve una

variable de tipo “long long” y recibe dos parámetros de entrada; una variable “long long” con el valor de la potencia cuadrada de la muestra, y una variable “long long” que es la que contiene la información de las muestras anteriores.

```
long long FiltroIIR(long long MuestraEntrante, long long PromedioAnterior)
{
  long long Resultado = 0LL;

  Resultado = MuestraEntrante -  PromedioAnterior;
  Resultado = Resultado >> 10;
  Resultado += PromedioAnterior;

  return(Resultado);
}
```

La operación consiste en restar el valor de la muestra entrante a la variable “PromedioAnterior”, desplazar el resultado 10 bits a la derecha (que es lo mismo que multiplicar por 2^{-10} el contenido) y sumar el valor de “PromedioAnterior”. Finalmente se devuelve el resultado.

El proceso en tiempo real inicia reiniciando la variable de conteo de muestras, y habilitando las interrupciones (que determinarán cuando hay una muestra disponible. A continuación, un bucle asegurará que se procesaran tantas muestras como configuremos. Dentro de este bucle hay una estructura de selección, la cual analiza la marca “WSMP”, que cuando esta a “HIGH” indica que hay una muestra disponible. En el caso de que la haya, se lee la muestra, se adapta a la variable de Arduino, se eleva al cuadrado, se filtra, se incrementa el contador y se desactiva la marca WSMP de disponibilidad de muestras. A partir de aquí comenzaría de nuevo el proceso.

```
long long ProcesoTiempoReal ()
{
  int ContadorDeMuestras = 0;
  long long Muestra = 0LL;
  long long Promedio = 0LL;

  WSMP = LOW;
     attachInterrupt(0,Interrupcion,FALLING);
     while (ContadorDeMuestras <= 4000)
     {
       if (WSMP == HIGH)
       {
             Muestra = LeerRegistro(WAVEFORM,3);
             Muestra = AdaptarMuestra(Muestra);
             Muestra *= Muestra;
             Promedio = FiltroIIR(Muestra, Promedio);
             ContadorDeMuestras++;
             WSMP = LOW;
       }
     }
     detachInterrupt(0);
     return(Promedio);
}
```

Para saber cuándo hay una muestra disponible, se hace uso de una interrupción externa, cuando la salida de solicitud de interrupción pase a nivel bajo, Arduino irá directamente a la rutina del servicio de interrupciones:

```
void Interrupcion ()
{
    int ESTADO = 0L;
    ESTADO = LeerRegistro(RSTSTATUS, 1);
    WSMP = bitRead(ESTADO, 3);
}
```

Esta rutina lee el contenido del registro de interrupciones, si la interrupción es debido a que hay una muestra disponible, configurará la marca "WSMP" a nivel alto.

- **Post-Procesamiento**

El post-procesamiento consiste en convertir el código obtenido en el valor real que se ha medido. Para ello se usa una función que recibe como parámetro el código obtenido en el proceso en tiempo real. Esta función transforma este a una variable real, y opera este resultado para aplicar la función de transferencia de los ADC:

```
float ConvertirTension()
{
   float Resultado = 0.0f;
   Resultado = float(PromedioTension);
   Resultado *= (2.39)*(1.05);
   Resultado /= (2048*3.0492);
   return(Resultado);
}
```

El algoritmo principal del post-procesamiento consiste en hacer la raíz cuadrada del promedio obtenido, y aplicar la función anterior para convertir este código en un valor real, y mostrarlo por la pantalla del ordenador.

```
void PostProcesamiento (long long CodigoPTR)
{
   CodigoPTR = sqrt(CodigoPTR);
   Serial.println(ConvertirCodigo(CodigoPTR),DEC);
}
```

- **Algoritmo principal**

El algoritmo principal para el cálculo RMS usa 5 líneas de Arduino: 4 líneas para la comunicación SPI (DIN, DOUT, CHIPSELECT Y RELOJ) y otra línea para la línea de solicitud de información. También se ha almacenado el valor de los registros que se van a usar y se declara una variable lógica, que será la marca para indicar que hay una nueva muestra disponible. El bucle principal está continuamente configurando el modo de muestreo de onda en ADE7759 y realizando el cálculo RMS

```
#include <SPI.h>

enum
{
      ENTRADA       = 11
    , SALIDA        = 12
    , RELOJSPI      = 13
    , CHIPSELECT    = 10
    , IRQ           = 2
};

enum
{
      WAVEFORM      = 0x01
    , RSTSTATUS     = 0x05
    , MODE          = 0x06
};

volatile bool WSMP = LOW;
long Muestra = 0;
long long MuestraCuadrado = 0LL;
long long PromedioTension = 0LL;
float TensionT            = 0.0f;


void setup()
{
  pinMode(ENTRADA, OUTPUT);
  pinMode(RELOJSPI, OUTPUT);
  pinMode(CHIPSELECT, OUTPUT);
  pinMode(IRQ, INPUT);
  pinMode(SALIDA, INPUT);

  digitalWrite(CHIPSELECT, HIGH);
  digitalWrite(IRQ, HIGH);

  SPCR = ((1 << SPE)|(1 << MSTR)|(1 << CPHA));

  Serial.begin(19200);
}
void loop()
{
  EscribirRegistro(MODE,0x780C,2);
  ProcesoTiempoReal();
  PostProcesamiento();
}
```

- **Experiencia**

Para comprobar la linealidad del funcionamiento, se ha realizado un test al dispositivo en un laboratorio de electricidad. Se ha usado el código descrito anteriormente "TEST_RMS.PDE" para implementar la medida de valores RMS, y se ha configurado para medir la señal de tensión, en el canal 2.

Dado que la tensión de red no es estable, se ha hecho una modificación al código para que no se produzcan variaciones muy grandes en el código de salida. En un principio, se ha diseñado el algoritmo con una sensibilidad de 10 mV por estado. Para evitar estas fluctuaciones se ha modificado la sensibilidad, y se han usado 12 bits por muestra, lo que significa que la sensibilidad pasa a ser de 0,198 voltios por estado.

$$\text{Sensibilidad} = \underbrace{\frac{2{,}42}{\left(\frac{262144}{2^7}\cdot 3{,}0492\right)}}_{\text{Funcion de transferencia del ADC}} \cdot \underbrace{511}_{\text{Atenuación}} = 0{,}198 \text{ Voltios/Estado}$$

El canal de tensión incorpora un filtro paso-bajo LPF1 que atenuará la señal 5%, lo que significa que la sensibilidad será de aproximadamente 0,208Voltios/Estado:

$$|H(f)| = \frac{1}{\sqrt{1+\left(\frac{50\text{ Hz}}{156\text{ Hz}}\right)^2}} = 0{,}95$$

$$\text{Sensibilidad} = 0{,}198 \cdot 1{,}05 = 0{,}208 \text{ Voltios/Estado}$$

Se ha conectado la entrada del canal de medida de tensión del medidor a un reóstato, que está conectado a un transformador en serie con una resistencia. Modificando el valor del reóstato se modifica el valor de tensión. De esta forma se han tomado medidas en diferentes puntos. La siguiente tabla muestra los resultados:

Tensión de entrada (V)	Código Máximo	Código Mínimo	Valor medio
120	571	573	572
125	593	595	594
130	618	621	620
135	639	641	640
140	663	667	665
145	689	686	688
150	715	713	714
155	740	737	739
160	763	760	762
165	785	783	784
170	809	805	807
175	829	831	830
180	859	855	857
185	880	878	879
190	904	900	902
195	929	922	926
200	954	948	951
205	979	973	976
210	1002	998	1000
215	1028	1021	1023
220	1051	1048	1050
225	1075	1069	1072
230	1098	1090	1094
235	1122	1116	1119
240	1144	1139	1142
245	1170	1163	1167
250	1196	1188	1192
255	1217	1209	1213
260	1241	1237	1239

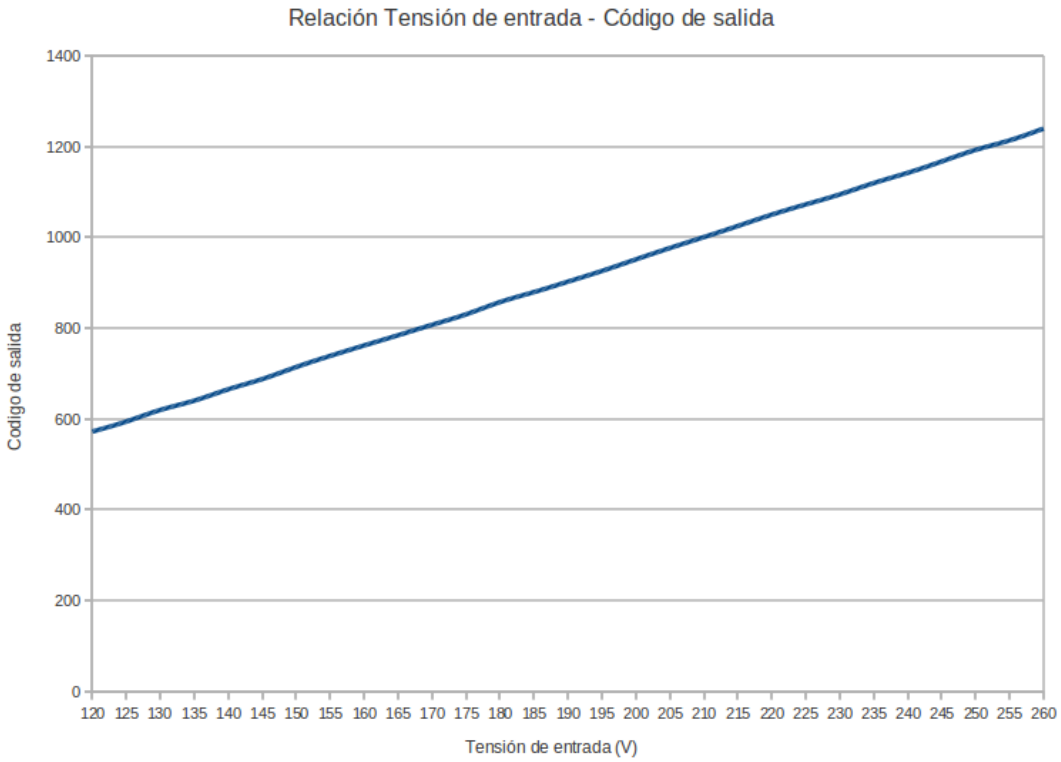


Imagen 5.5.2 – Característica de salida del canal de tensión

Se han calculado los coeficientes de la recta de regresión que genera el menor error, y los resultados obtenidos son los siguientes:

$$f(Codigo) = Codigo \cdot 0{,}2095 + 0{,}615$$

Se puede observar que la pendiente obtenida es prácticamente igual que la sensibilidad que se planteo inicialmente (0,208 voltios por estado). Además existe un error de offset en la medida de -0,615 Voltios, que son aproximadamente 3 estados. La siguiente imagen muestra el error cometido en la medida si se aplica la recta de regresión anterior:

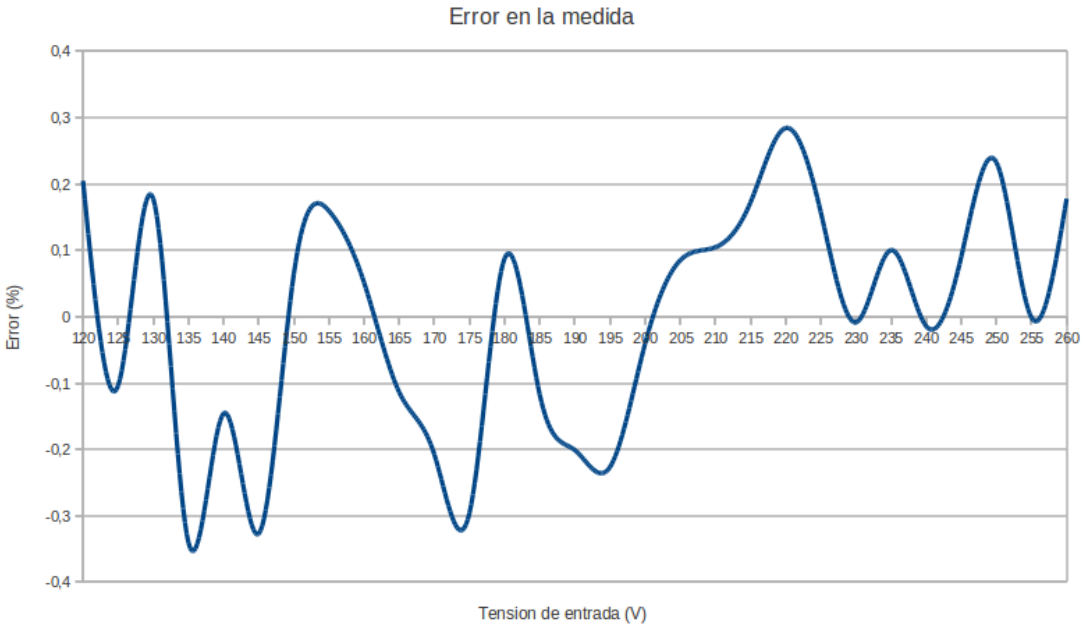


Imagen 5.5.3 – Error cometido en la medida

Se puede observar como el mayor error cometido es de 0,3%. Hay que tener en cuenta los errores introducidos en el proceso de medida como consecuencia de las fluctuaciones de la tensión de red. Aún así, se puede comprobar cómo la salida es completamente lineal.

5.6. IMPLEMENTACIÓN DEL MODO DE ACUMULACIÓN DE ENERGÍA, CALCULO DE POTENCIA APARENTE Y FACTOR DE POTENCIA

El cálculo de la potencia aparente y el factor de potencia se implementan con el procedimiento de cálculo de valores eficaces descrito anteriormente. Suponiendo que el dispositivo de medida está calibrado y no tenemos errores de offset, la potencia aparente se calcula a partir del valor de corriente eficaz y de tensión eficaz:

```
void CalculoRMS(int Opcion)
{
  long long Promedio = 0LL;
  Promedio = ProcesamientoTiempoReal (Opcion);
  PostProcesamiento (Promedio, Opcion);

}

void CalculoPotenciaAparente()
{
  long long PromedioCorriente = 0LL;
  long long PromedioTension = 0LL;
  long long PotenciaAparente = 0LL;
  float PotenciaAparenteReal = 0.0f;

  PromedioCorriente = ProcesamientoTiempoReal(1);
  PromedioCorriente = sqrt(PromedioCorriente);
  PromedioTension = ProcesamientoTiempoReal(2);
  PromedioTension = sqrt(PromedioTension);

  PotenciaAparente = (PromedioTension*PromedioCorriente);
  PotenciaAparenteReal = ConvertirValor(PotenciaAparente, 4);

  MostrarValor(PotenciaAparenteReal, 5);
}
```

Se puede ver que el algoritmo consiste simplemente en calcular la corriente eficaz y después la tensión eficaz, multiplicarlos y mostrar los datos. El cálculo del factor de potencia consiste en el mismo procedimiento, pero calculando también la potencia activa para finalmente dividir la potencia activa entre la potencia aparente:

```
void CalculoFactorDePotencia()
{
  long long PromedioCorriente = 0LL;
  long long PromedioTension = 0LL;
  long long PromedioPotencia = 0LL;
  float Corriente = 0.0f;
  float Tension = 0.0f;
  float PotenciaActiva = 0.0f;
  float PotenciaAparente = 0.0f;
  float FP = 0.0f;

  PromedioCorriente = ProcesamientoTiempoReal(1);
  PromedioCorriente = sqrt(PromedioCorriente);
  PromedioTension = ProcesamientoTiempoReal(2);
  PromedioTension = sqrt(PromedioTension);
  PromedioPotencia = ProcesamientoTiempoReal(3);

  Corriente = float(PromedioCorriente);
  Tension = float(PromedioTension);
  PotenciaActiva = float (PromedioPotencia);
  Corriente *= 0.00818;  // Coeficiente calculado en cal.
  Tension *= 0.00649;    // Coeficiente calculado en cal.
  PotenciaActiva *= 0.0256;  // Coeficiente calculado en cal.
  PotenciaAparente = Corriente*Tension;
  FP = PotenciaActiva/PotenciaAparente;

  MostrarValor(FP,6);
}
```

El procedimiento de acumulación de energía se describió anteriormente en el apartado 4.7. El funcionamiento consiste básicamente en establecer el modo de acumulación de energía y habilitar el registro de interrupción la marca CYCEND para que se produzca una interrupción cuando se acabe el proceso de acumulación. Una vez que esto suceda, se leerá la energía que se ha acumulado en el registro de acumulación de energía con reset, y mientras este comienza de nuevo a acumular energía se realizará lo que en el cálculo RMS se denomina el postprocesamiento, que en este caso consiste en pasa el valor de un código a Kilojulios. A continuación se muestra el código programado para la función acumular energía, la cual devuelve el valor de Kilojulios acumulados en 200 semiperiodos de la señal de entrada:

```
float AcumularEnergia ()
{
     long long Muestra = 0LL;
     float EnergiaConsumida = 0.0f;
     FINCYC = LOW;

     attachInterrupt(0,Interrupcion,FALLING);
     while (FINCYC == LOW)
     {
     }
     detachInterrupt(0);

     Muestra = LeerRegistro(RSTENERGY,5);
     Muestra = AdaptarMuestra(Muestra,4);

     EnergiaConsumida = float(Muestra);
     EnergiaConsumida *= (0.04873*2); //Conversion a Watios
     EnergiaConsumida /= (3579545);
     EnergiaConsumida /= 3600;        //Conversion a kJulios(kW por hora) consumidos
     EnergiaConsumida /= 1000;
     return (EnergiaConsumida);

}
```

Para comprobar el funcionamiento se ha conectado una carga resistiva (una estufa eléctrica) con dos niveles; en uno consume aproximadamente 740W y en otro 1400W. A continuación se muestran los resultados obtenidos:

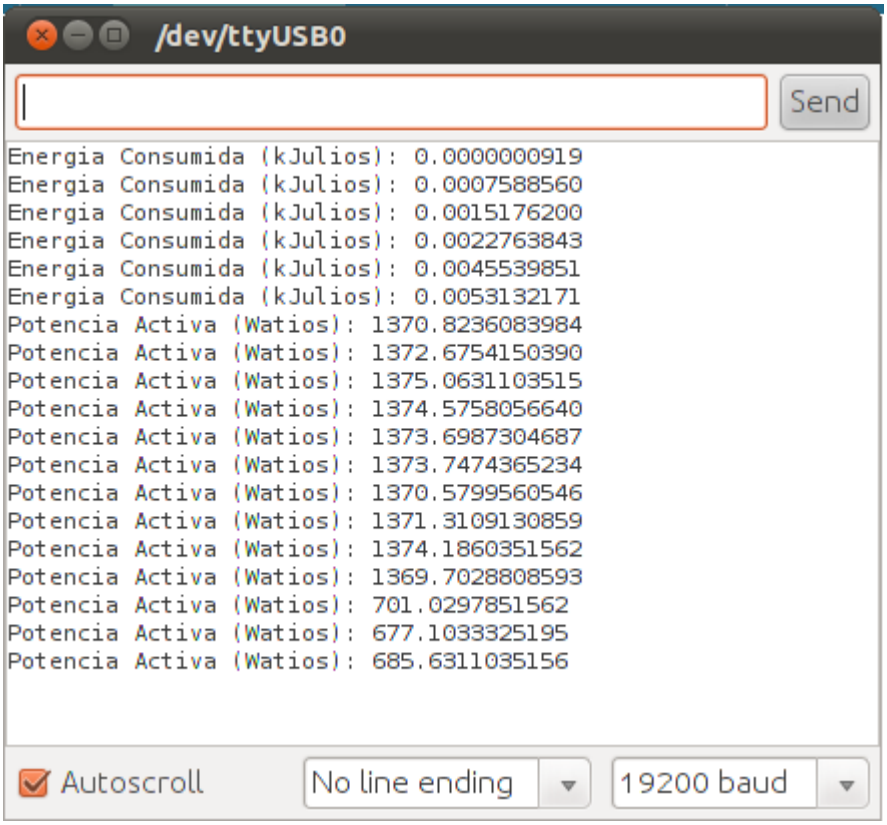


Imagen 5.6. – Imagen del resultado del cálculo de potencia activa

Imagen 5.6.1 – Resultado del cálculo del factor de potencia

En esta imagen se puede observar como el cálculo del factor de potencia desvela un desfase de 0,05º, ya que la carga empleada es resistiva.

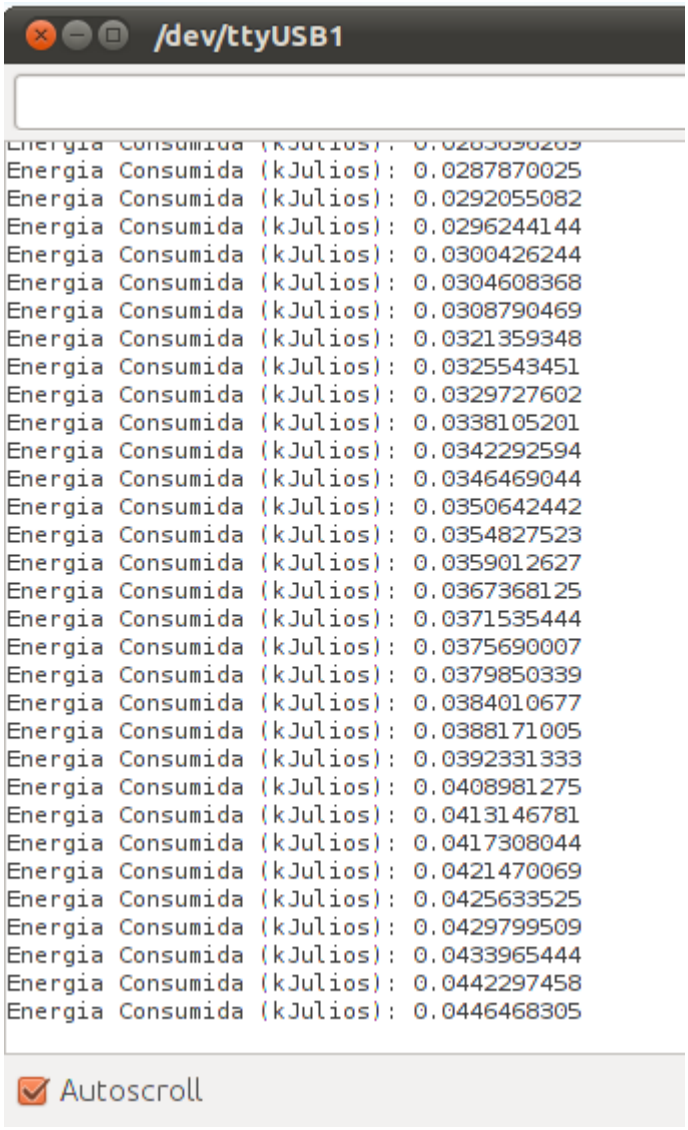


Imagen 5.6.3 – Calculo de energía consumida

5.7. IMPLEMENTACIÓN DEL ALGORITMO DEL MEDIDOR

El código del algoritmo se incluye en el anexo 1. Debido a la gran cantidad de líneas, se ha dividido en subprogramas para facilitar la legibilidad por otros usuarios. El criterio establecido a la hora de separar cada parte ha sido la ordenar el código por la función que realizan: el algoritmo principal, la comunicación SPI, los mensajes por el LCS, el servicio de interrupciones, las operaciones con las muestras, las rutinas de configuración del modo de operación de ADE7759, y los procedimientos de operación. Todos los subprogramas empleados han sido descritos previamente en apartados anteriores, si bien se han realizado algunas modificaciones para poder combinarlos todos. La implementación de los componentes también esta descrita (por partes) en los apartados anteriores. A continuación se describen las diferencias más notables en cada bloque del programa;

El algoritmo principal comienza con la declaración de variables (conexiones de Arduino, direcciones de los registros, marcas de habilitación, variables para algunas operaciones). Destacar la conexión de los botones y del LCD, ya que se han cambiado de posición respecto del usado en los apartados anteriores. Se pueden observar dos subprogramas; **RealizarAccion()** y **MostrarOpcion().** Estos subprogramas sirven para mostrar los mensajes en el display durante el proceso de selección de la operación a realizar por parte del usuario, y para realizar la función cuando sea seleccionada. El bucle del programa consiste en un algoritmo como el descrito en el apartado 5.3 Implementación del menú de selección. Cuando se pulsa el botón ATRÁS, se realiza un "reset" a ADE7759 para deshabilitar el modo de funcionamiento que esté realizando.

La comunicación SPI es un algoritmo idéntico al descrito en el apartado 5.4 Implementación del bus SPI. En el bloque de mensajes LCD se encuentran todos los mensajes que se pueden mostrar durante la operación del medidor. A cada uno le corresponde un nombre, y en el caso de tener que mostrar un resultado, obtiene el valor de este como parámetro.

El bloque de operaciones con muestras contiene los subprogramas necesarios para el procesamiento en tiempo real, para convertir los valores y del procesamiento en

tiempo real y post-procesamiento. Existe el problema que el mismo subprograma se usa para distintos cálculos, como por ejemplo, el procedimiento para adaptar las muestras a la variable de Arduino; en el caso de la tensión las muestras tienen 17 bis, y las de corriente 12 bits, y las muestras que se leen desde el registro de acumulación de energía tienen 40. La solución es crear un parámetro de selección, de manera que si se pasa un 1 en este parámetro para adaptar la muestra, realizará la acción para la muestra de corriente, y si se le pasa un 2 adaptará la muestra de tensión. Esta estrategia se ha utilizado para todos los procedimientos que sirven para diferentes funciones.

La rutina de configuración del modo de operación incluye los procedimientos para escribir en ADE7759 los registros necesarios para iniciar una operación, por ejemplo la de acumulación de energía. En los procedimientos de operación se encuentran los procedimientos descritos en los apartados 5.5 y 5.6 para el cálculo de los parámetros a medir.

.

6. DISCUSIÓN Y CONCLUSIONES

En este trabajo se han expuesto los métodos llevados a cabo para diseñar e implementar un medidor de diferentes parámetros de energía eléctrica mediante el circuito integrado ADE7759 con una placa Arduino. Las experiencias realizadas durante la redacción de este trabajo se han realizado con una placa Arduino Nano, que es la mejor opción para trabajar en con placas de prototipado. Si bien este tipo de soporte no es lo más recomendable para aplicaciones de digitalización, las experiencias realizadas han servido para comprobar la posibilidad de pasar a la siguiente fase en el diseño del dispositivo, que consistiría en la fabricación de una placa de circuito impreso para los componentes. Es recomendable considerar dos posibilidades muy interesantes; una es la de implementar el sistema en una placa de circuito impreso con las medidas de una placa Arduino como Arduino Duemilanove, de manera que se crearía lo que se llama una shield. Otra posibilidad es la de utiliza Arduino Nano y diseñar la placa de modo que puede ser introducido en unos zócalos fácilmente para poder usarlo en otros proyectos.

El sistema ofrece los resultados tanto por la pantalla del PC como por el display LCD que tiene el circuito implementado. El motivo por el cual se usa el display es para que se pueda usa el dispositivo sin un ordenador, aunque realmente el uso del puerto serie añade gran potencialidad para la visualización de datos. Para implementar una interfaz gráfica es recomendable utilizar Processing.

Arduino ofrece la posibilidad tanto de mostrar simplemente los resultados, como actuar sobre otros dispositivos en función de las medidas realizadas. Por ello, el medidor diseñado puede ser de gran utilidad en aplicaciones domésticas para el control del gasto energético, como por ejemplo el consumo energético de un calefactor. El rango de medida del dispositivo está determinado por la limitación del sensor en el caso de la corriente, y del ruido de fondo en la conversión para la señal de tensión. Redimensionando las redes de entrada (divisor de tensión y sensor de corriente) se puede modificar el rango de medida de los parámetros. Con algunas modificaciones, el sistema puede realizar medidas de señales continuas.

El dispositivo final tiene un coste aproximado de 17,715€ considerando los componentes y sin contar con Arduino. Esto es porque si se tiene en cuenta que Arduino puede programarse fácilmente y ser usado en otros proyectos, es interesante que el medidor se diseñe con vistas a ser una aplicación para una placa Arduino, y así poder usar una misma placa en múltiples proyectos. El precio de una placa Arduino puede variar desde los 20€ de una placa Arduino UNO hasta 42€ que cuesta un Arduino Nano.

En cuanto a las medidas realizadas, se ha comprobado como con el algoritmo diseñado se pueden medir valores eficaces en las señales de entrada con un error máximo del 0,3%. Un factor que es muy importante a la hora de determinar la precisión de la medida es la tensión de referencia de los convertidores. Para las pruebas realizadas se ha utilizado la tensión de referencia interna de ADE7759. El problema que supone esto es que varía muy fácilmente con la temperatura, y esto hace que los coeficientes de calibración se modifiquen, introduciendo por lo tanto un error en la medida. Por esto, es recomendable incluir un regulador de tensión para que no varíen estos coeficientes.

El algoritmo implementado también mide potencia activa, potencia aparente, factor de potencia y energía consumida de una forma satisfactoria. Todos los algoritmos implementados han funcionado correctamente, excepto el cálculo del factor de potencia y de potencia aparente, los cuales muestran saltos en las medidas. El comportamiento parece indicar que hay algún conflicto en la selección de modos de operación de ADE7759.

Aunque no se ha podido comprobar los errores que se cometen en el cálculo de los parámetros anteriores, los resultados indican que el medidor presenta un comportamiento lineal, y por lo tanto es viable la fabricación del dispositivo.

7. BIBLIOGRAFÍA

- Artículo sobre comunicación SPI - Fuente: Wikipedia: http://es.wikipedia.org/wiki/Serial_Peripheral_Interface
- Tutorial sobre pulsadores en Arduino – Fuente: Adafruit http://www.ladyada.net/learn/arduino/lesson5.html
- Datasheet ADE7759 – Fuente: Analog Devices
- Foro de ayuda de Arduino: http://arduino.cc/forum/index.php/board,32.0.html
- "RMS calculation for energy meter applications using ADE7756", Notas de aplicación Analog Devices AN-578
- "A power meter reference design base don ADE7756", Notas de aplicación Analog Devices AN-564
- Artículo sobre resistencia de pull-up, Fuente: Wikipedia, http://en.wikipedia.org/wiki/Pull-up_resistor
- Página de Arduino: http://arduino.cc/
- Página de Fritzing: www.fritzing.org
- Artículo sobre Filtros digitales. Fuente: Wikipedia, http://es.wikipedia.org/wiki/Filtro_digital
- Documentación técnica de Arduino, página de Arduino